SPi	Journal Code				Article ID				Dispatch: 26.05.15	CE: Sarah Jane T. Jalandoni
	S	A	T		1	1	2	0	No. of Pages: 19	ME:

INTERNATIONAL JOURNAL OF SATELLITE COMMUNICATIONS AND NETWORKING
Int. J. Satell. Commun. Network. (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sat.1120

A deep analysis on future web technologies and protocols over broadband GEO satellite networks‡

Luca Caviglione[1], Nedo Celandroni[2], Matteo Collina[3], Haitham Cruickshank[4], Gorry Fairhurst[5], Erina Ferro[2,*,†], Alberto Gotta[2], Michele Luglio[6], Cesare Roseti[6], Ahmed Abdel Salam[6], Raffaello Secchi[5], Zhili Sun[4] and Alessandro Vanelli Coralli[3]

[1]*National Research Council (CNR), Institute of Intelligent Systems for Automation (ISSIA), Genoa, Italy*
[2]*National Research Council (CNR), Institute of Information Science and Technologies (ISTI), Pisa, Italy*
[3]*University of Bologna, Bologna, Italy*
[4]*University of Surrey, Guilford, UK*
[5]*University of Aberdeen, Aberdeen, UK*
[6]*University of Roma Tor Vergata, Rome, Italy*

SUMMARY

The goal of this work was to understand the direction of the emerging web technologies and to evaluate their expected impact on satellite networking.

Different aspects have been analysed using both real satellite testbeds and emulation platforms in different test sites in Europe. This analysis included an evaluation of those HTTP/2.0 specifications, which were implemented and released as open-source code in the experimental release of the SPDY protocol. SPDY performance was evaluated over satellite testbeds in order to understand the expected interaction with performance-enhancing proxies (including scenarios with a SPDY proxy at a satellite gateway), the impact of security and the effect of satellite capacity allocation mechanisms. The analysis also considered the impact of application protocols and the delay induced by end-system networks, such as a satellite-connected WiFi network. Copyright © 2015 John Wiley & Sons, Ltd.

Received 7 August 2014; Revised 30 April 2015; Accepted 30 April 2015

KEY WORDS: satellite; SPDY; HTTP; web; WebSocket

1. INTRODUCTION

In the last 10 years, the types of data exchanged using HTTP [1] have radically changed. Early web pages consisted of a few tens of kilobytes of data and were normally not frequently updated (static web). Today, typical web pages are far more complex, consisting of tens of elements, including images, style sheets, programming scripts, audio/video clips and HTML frames [2]. Moreover, many web pages are updated in real time (e.g. when they are linked to live events or dynamic databases). Web interfaces are used in a wide range of non-browsing applications, for example, to interact in real time with distributed web applications, where both data and application reside on the remote side.

Since today, HTTP/1.1 is called to operate in a context different from the one it was originally proposed, and some of the limitations of the HTTP model have become apparent. The original HTTP request/response model, where each web object is singularly requested, has been shown to not scale to modern complex web pages and introduces a significant amount of overhead at start-up. Also, HTTP has poor cross-layer interaction with the lower network layers (in particular the transmission control

*Correspondence to: Ferro Erina, National Research Council (CNR), Institute of Information Science and Technologies (ISTI), Pisa, Italy.
†E-mail: erina.ferro@isti.cnr.it
‡Work co-funded by European Space Agency (ESA) in the framework of the ESA support to the SatNEx III Network of Experts, CoO3 'State-of-the-art Web Technologies and Protocols', ESA Contract 23089/10/NL/CLP.

protocol (TCP) [3]), contributing to poor responsiveness for web applications. All of this occurred at a time when changing patterns of web use have raised concerns on performance.

Several initiatives from private industry and standardisation bodies recently started a review of HTTP and the related protocol stack to make it more suited to the new web. SPDY (speedy protocol) [4], quick user datagram protocol Internet connections [5] and Minion [6] are prototypal protocol implementations from different vendors that seek to make web applications more responsive, more network friendly and more suitable to the new web design. These experimental protocols have been taken by the Internet Engineering Task Force (IETF) and World Wide Web Consortium (W3C) as an input to the definition of HTTP/2 [7], a new standard to improve HTTP performance using the experience cumulated in the last years with web traffic.

- The security model in HTTP/2~~/SPDY~~ is different from that in HTTP/1.1. Authentication and encryption are considered key elements both to protect users from malicious servers and to guarantee user privacy. This means that future web services will increasingly make use of security mechanisms, such as transport-layer security (TLS) [8] or secure socket layer (SSL) [9], to protect their users. As a consequence, satellite operators will experience increasingly more difficulties in accessing HTTP headers to enforce policies or implement performance-enhancing proxies (PEPs) [10], such as HTTP accelerators.
- Changes at the transport level can also impact the performance of HTTP. There is now a class of proposals for TCP modification designed to boost TCP performance at start-up. Some of these modifications, such as TCP Fast Open (TFO) [11], require new TCP options. Satellite equipment needs also to evolve to be able to decode the new TCP headers; otherwise, it may fail to benefit from the new performance improvements or even impair connectivity. HTTP/2~~/SPDY~~ tries to multiplex as much as web page data as possible over a single TCP transport connection. This not only reduces the overhead of a HTTP stream but also improves the network friendliness of HTTP streams towards other traffic, avoiding disruptions in their performance. The use of a single connection has both pros and cons for satellite. On the one hand, fewer longer-lived flows offer good performance requiring less state in the satellite system (especially smaller binding tables when network address/port translation is used). On the other hand, fewer flows could be more sensitive to the loss or delay of packets resulting in increased latency over poor quality links.
- HTTP/2~~/SPDY~~ can simplify the design of proxies. HTTP/2~~/SPDY~~ takes care of data multiplexing, data compression and security, thus greatly simplifying the job of HTTP accelerators, which otherwise had to implement translators to perform many of these functions. In addition, HTTP/2~~/SPDY~~ regulates the HTTP flow rate, taking into account the processing speeds of intermediary nodes.

However, the change in security and transport models needs to be considered as one of the updates that may demand changes in the satellite equipment. This need to update the equipment in order to follow changes in network protocols is not new. To avoid ossification (where parts of the network fail to support new performance improvements), satellite operators already need to be careful to update configurations/software as new network technologies emerge. Before exploring how to update a PEP, it is important to understand whether and how HTTP/2~~/SPDY~~ will benefit from satellite-specific protocol acceleration.

This paper reports experimental results of a series of measurement campaigns with ~~a~~ set of features included in HTTP/2 ~~but derived~~ and implemented in SPDY. By now, we will refer to SPDY as the reference architecture and the experimental implementation of such features. SPDY protocol has been experimentally tested over satellite and compared with the present web architecture based on HTTP/1.1. Such trials were conducted in the context of SatNex III NoE (http://www.satnex.org).

In general, SPDY benefits the performance of web browsing via satellite by reducing latency and significantly lowering the overhead for several types of web pages. In particular, multiplexing a sequence of small web objects on a web page onto a single connection enables significant gain and effectively reduces the HTTP overhead per object and per connection. The server push feature of SPDY adds the ability for the server to send objects without explicit request and was found particularly useful to reduce the download time. However, we also observed places where the tested

DOI: 10.1002/sat

implementations did not perform optimally and where performance over satellite-connected WiFi networks was comparable with HTTP/1.1.

This paper summarises the experience with ~~HTTP/2/~~SPDY in the SatNex project and discusses benefits and challenges posed by the new techniques for web content delivery and their impact to the satellite networking. The paper introduces some of the novelties of HTTP/2, the new connection-oriented framework for web applications and the transport-layer enhancements, respectively, in Sections 2, 3 and 4. The experimental test setups are described in Section 5. Results are presented, respectively, in Sections 6, 7 and 8. In particular, Section 6 represents a performance comparison between HTTP/1.1 and SPDY over the satellite link, Section 7 evaluates SPDY performance with respect to different bandwidth on demand (BoD) mechanisms and Section 8 shows results with real-time flows over satellite. A discussion on the implications of using SPDY is given in Section 9. Finally, Section 10 concludes the paper with recommendations for satellite operators and web protocol designers.

2. AN INTRODUCTION TO HTTP/2

It has been recognised that as network bandwidth increases, the greater obstacle to the performance of web technology is latency rather than scarcity of capacity. HTTP/2 [7] is a new protocol being design by the IETF and W3C to address the limitations and lack of flexibility of the currently widely deployed HTTP/1.1 [1]. The origin of the emerging HTTP/2 standard may be traced back to a Google™ (Google Inc., Mountain View, CA, USA) suite of protocols, called SPDY. Proposed modifications to HTTP/1.1 include suppression of unnecessary round trip time (RTT) delays to request and deliver web objects, multiplexing of web streams into a single connection, removal of the head of line blocking, more compact encoding of HTTP headers, server-initiated transmissions (server push), a new security model and an extensible design.

HTTP/2 is expected to supersede HTTP/1.1 in all the user cases where HTTP/1.1 is currently employed. At the present time, the scope and content of the standards are being developed within the relevant working groups (WGs). There are, as yet, no formal standards, so this analysis is based on the available material only~~, that is, based on SPDY~~.

2.1. Characteristics of SPDY

The current HTTP specifications rely on multiple parallel connections to expedite web page object download. In particular, HTTP/1.1 defines a transport mode called HTTP persistence, where the same TCP connection can be reused multiple times for subsequent HTTP request/response transactions.

In recent years, the process of parallelisation of TCP flows carrying HTTP data has gone to the extremes. Domain sharding is a well-known practice to share content server load between multiple sites. It is sometimes used as a trick to induce browsers to open more parallel connections than they normally should to one site. The HTTP/1.1 specifications recommend using at most two connections per domain. However, browsers more frequently use four to six connections per domain [1]. Clearly, if objects are split across several domains, browser can open connections up to the number of domain multiplied by the number of connections per domain. Thus, browser can open tens of connections for the same web page.

Domain sharding became necessary to compensate for the huge increase in the number of objects per page. This figure has more than tripled in the past decade, now getting close to the hundreds of objects per page [12]. However, the multiplication of TCP connections per page has serious consequences:

- First, the aggregate of connections behaves aggressively with respect to other traffic (i.e. it behaves as an aggregate of TCP flow rather than a single flow).
- Second, it increases the per-connection overhead, including the signalling to set up a connection and start-up latency.
- Moreover, sharding increases the per-connection load to the network. This is an issue for intermediaries that need to monitor user traffic or track connection state, such as gateway or proxy, and need to track up hundreds of connections per user. Fewer longer-lived flows offer better performance anyway and require much less state in the satellite system (especially smaller binding tables when network address port translation is used).

DOI: 10.1002/sat

The answer of SPDY to this problem is to adopt the 'Single Connection' paradigm. A single connection per domain ~~should be~~ sufficient to transport all the data for a web page, as long as multiple objects can be multiplexed onto the same flow. If a stream identifier is associated with a message, the server can insert more small messages into the same response. This reduces sensibly the packet overhead. In addition, this mitigates the 'head of line blocking' problem with persistent HTTP/1.1, in which more important objects for visualisation may be queued behind blocked and less significant ones to the same TCP connections.

The gain in multiplexing and prioritising objects of a web page is clearly dependent on the type of page. Web pages more fragmented will probably benefit more of serialising object into the same connection. However, the benefits of SPDY can be significant. Figure 1 shows the average completion time of the flag test web page, a sequence of small icons referred by a same index web page (see Section 5.1 for the details of the experimental setup). This experiment clearly highlights that the reduction in web page download time over satellite can be significant. F1

2.2. SPDY prioritisation mechanisms

Introducing message identifiers in SPDY allows servers to implement a series of techniques. In reality, SPDY defines a hierarchy of identifiers differentiating between stream and requests. Roughly, requests map to specific web resources, and streams are containers for requests of similar type. This gives web servers higher flexibility to cover a wider range of use cases:

- While HTTP/1.1 decreases the web page download time by using multiple connections, high-priority resources might be delivered slowly because of concurrent delivery with some non-critical resource. SPDY request prioritisation allows a server to establish which resources are more important than others so to deliver them first over the same connection. The client may also help the server to take the right decision by indicating the objects that it is able to receive and process.
- SPDY offers a stream priority mechanism allowing the server to specify the relative priority of each stream. Stream priority is used to inform the peer which streams are more important than

Colour online, B&W in print

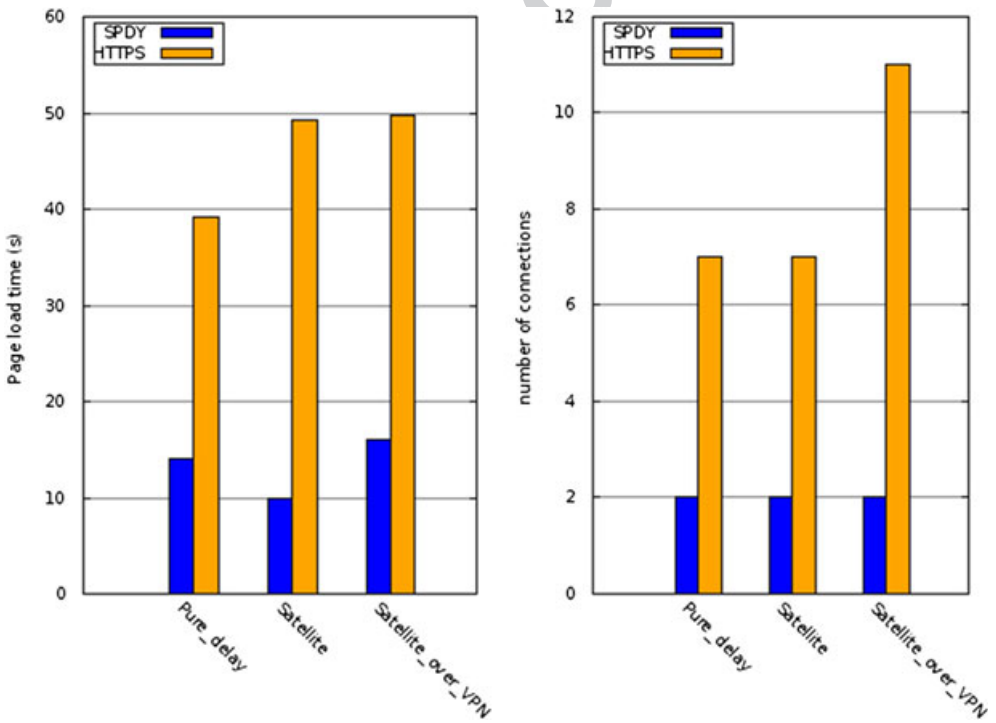


Figure 1. Multiplexing gain and number of connections with SPDY for the flag test web page. VPN, virtual private network.

DOI: 10.1002/sat

others. In some cases, the browser may need to specify a preferential order for web objects if it cannot process some resources before others are received.

The prioritisation mechanism aims to improve performance by anticipating the transmission of the most critical objects and defining critical paths. A client can ensure that resources needed immediately do not compete for bandwidth capacity with less important resources transfer. Requested resources with a lower priority are queued waiting to fill any spare bandwidth with useful data. This scheduling mechanism was not possible in HTTP/1.1 where any request, once issued, could not be rescheduled.

Effective scheduling of web resources is a complex optimisation task that takes into account the structure of a web page. SPDY offers a way to indicate this structure. In other words, priorities can be used by each peer to signal the dependencies in a web page and receive the web objects in the correct order.

2.3. Server push of HTTP objects

HTTP/1.1 servers can send resources to the client only when they are explicitly requested. However, in most of the cases, the server can anticipate which resources the client is going to request. For example, if the client has requested an index page, it is very likely that it will request the style sheets and the code associated with that page the next RTT.

SPDY proposes an optional server push that allows a server to deliver unrequested messages to the client. Server push may save precious RTTs as the server can deliver a resource along with all the associated resources. Pushing data from server to client is also useful in the context of dynamic web. Servers can update dynamically client without explicit requests from their side.

Server push is particularly beneficial for satellite network because of the larger RTT. Also, it fits well the characteristics of a satellite channel. For example, future use of server push method could be associated with the broadcast/multicast capabilities of a satellite forward link. In a scenario with many users accessing simultaneously (or within of a short time range) to the same web resource (e.g. Internet protocol television live event), browsers could receive pushed data from the server without explicit requests. Q8

The use of server push could be also used with proxy caches at the satellite gateway (or even at a terminal). This would avoid pre-fetching resources already pushed by other terminals over the same satellite network.

2.4. Open issues with HTTP/2

At the time of writing, the HTTP/2 specification is far from finished, and many points remain under discussion within the HTTPbis WG. This section highlights some presently open issues expected to impact on satellite networks.

Server push is still a debated topic within the WG. Although it can be used to anticipate client requests, it could also transmit data that the client is not able to receive or use. Several people fear that server push may lead to large amounts of undesired data to the client.

Because server push is a server-side optimisation, it assumes that a server has good knowledge of the network characteristics and user expectations. In any non-cabled environment, the path characteristics may be hard to predict a priori, being a function of many parameters (propagation conditions, system load, service-level agreement enforcement, etc), and therefore, different optimisations may be needed when a path includes a satellite, cellular or wireless network. The way in which these impact the design and usage of server push will need to be determined before the effect on satellite networks can be fully understood. Q9

In addition, major server sites are keen to avoid accumulating state for inactive connections—both because it consumed memory and because it reduces the ability to perform server load balancing.

Many satellite networks have deployed PEPs to perform TCP and HTTP/1.1 acceleration. These have often become regarded as essential components for satellite networks that support web access. In networking terms, these boxes are a proxy/middlebox that works as an intermediary between the content server and the client.

DOI: 10.1002/sat

Transport-layer PEPs (e.g. split TCP) have been shown to work effectively with SPDY and are hence expected to continue to work with HTTP/2. The main issue is that HTTP/2 has motivated changes in the transport protocols and that any future transport PEP needs to be able to accommodate these changes to avoid the pitfalls of ossification (i.e. downgrading the transport to an old version so that the PEP can accelerate the transport, whereas a native transport with no transport PEP would have achieved higher performance).

The role of the intermediary in HTTP/2 and in particular how SPDY interacts with HTTP/1.1 still does need to be specified. The use of session encryption (TLS) ensures that a SPDY client and server explicitly acknowledge the presence of any intermediary that is allowed access and manipulates data transmitted between the server and the client. This form of middlebox interception is a much-requested feature because many network administrators wish to protect their network and filter and/or disallow certain sites. However, defining the (trusted) interaction of HTTP behaviour with middleboxes creates a range of problems, such as how to ensure confidentiality while granting access to data to third parties and how to provide end-to-end control of a flow that needs to be operated from within the network.

In present satellite systems, the interception of application-layer protocols is a key function of application-layer PEPs and used by products such as TurboPage™ [13]. The use of intermediaries will need to be different, and this suggests that the currently combined role of applications and transport PEPs could be better viewed as two separate roles, with the possible use of a 'standard' configured proxy for web acceleration. Q10

Performance-enhancing proxies can also influence the quality of service requested in the lower-layer satellite service, defining how flows are mapped to lower-layer functions. Such PEPs need to be aware of flow traffic patterns/requirements. Present systems often rely on deep packet inspection. This could not be studied in the current project, but as the transport layer evolves (possibly away from TCP) and there is increased use of encryption (e.g. TLS used by SPDY) and automated compression, access to upper-layer protocol headers will become impossible at a PEP. Satellite networks therefore need to differentiate traffic using other mechanisms, such as support of differentiated services packet marking.

3. A NEW FRAMEWORK FOR S APPLICATIONS

HTTP/2 architecture should be flexible enough to support the wide variety of devices and applications currently supported by HTTP/1.1. Recently, also wireless mobile applications have started to rely heavily on web resources generating even more use cases and interests in HTTP. According to [14], the new category of traffic has dramatically increased, almost doubling every year over the last 2 years, and it is currently accounting for almost one-fourth of the total web traffic. Wireless devices, however, have much more limited capability than other end systems and require different functionalities. In other words, mobile web introduces constraints in web design that ultimately impact the traffic pattern seen on wire.

Wireless devices may generate considerably different traffic profiles than wired end hosts. For example, to save energy, bandwidth and the resources of mobile devices, web pages are often crafted to reduce the amount of wireless data and to fit the capabilities of small devices. Also, mobile applications seek to control the amount and the type of data received from the server to correspond their requirements. For example, mobile clients will try to inhibit 'push' notification from the server or enable particular services, such as location-based services, if they can support them.

To address mobile web-specific problems, such as session management over wireless links, the IETF proposed WebSocket (WS) [15]. WS guarantees the flexibility and control mobile application need. Unfortunately, a large part of deployed mobile operating system (OS) still lacks WS support. This is the case, for example, of the widespread Android-2™, which does not support WS, or Apple iOS-4™ (Apple, Inc., Cupertino, CA, USA), which supports partially compatible versions of WS. In addition, WS deployment is sometimes impeded by intermediaries that reject non-HTTP/1.1 conformant traffic. For example, some 3G network operators are known to employ transparent proxies that block WS traffic. Q11 Q12 Q13

When WS is not available, mobile devices need to resort to polling methods based on HTTP/1.1, such as Ajax Long Polling (ALP) technique [16]. The initial lack of support for WS was foreseen by the IETF WG that designed methods to probe for incompatible intermediaries.

DOI: 10.1002/sat

Another promising technology called server-sent events (SSE) [17] was defined by W3C to address the needs of web real-time applications. SSE is also a part of the HTML5 specifications. SEE defines both a Javascript application program interface and a data format to send data through a normal HTTP request. Unlike ALP, the response HTTP body is left open, so new events can be forwarded to the client. SSE is a backward compatible specification, and most browsers implement this.

Mobile OSes and software updates, however, are subject to faster upgrades than hardware, and a rapid spreading of the WS is expected in near future [18]. As the benefits of the new protocol become apparent, new updates and software become available. As a part of their activity, the IETF WG promoted actions to quickly report situations of failure to work out quickly the causes of hindrance. Moreover, libraries implementing WS at the application layer can be used for specific applications when the OS support is lacking.

Issues of mobile web are also relevant to the satellite community. A common scenario sees mobile devices connected to the Internet through a wireless-connected satellite terminal with a satellite return access link. Even though the terminals can be also identified as 'mobile', because the OS of a handheld device can be either Apple iOS™ or Google Android™, this common setup has a fixed satellite terminal. Understanding the problems of real-time web applications by means of functionality and performance test is therefore a critical investigation.

4. SPDY AND THE TRANSPORT LAYER

HTTP is normally layered on TCP using the reliable in-order socket interface. In practice, TCP is used with workarounds mostly because an application developer can rely on its presence in the majority of systems. In addition, TCP does not provide an adequate 'service model' capable of dealing with proxy or firewalls. Recent proposals, such as Apple's Minion [6] or Google's quick user datagram protocol Internet connections [5], try to address this issue, and there is likely to be further development at the IETF to define a service model to be used for HTTP transport.

All transport protocols need to perform congestion control to prevent overwhelming a network path. The effort to improve TCP congestion control and make it more responsive for low-latency applications is a part of the remit of the IETF TCPM WG. The problem of latency was attacked from two different sides. On the one hand, the WG is trying to standardise methods that allow a faster start of a connection, using Fast Open (TFO) [19] and larger Initial Window (IW10) [20], or quickly restart idle connections, using new CWV [21] in combination with ~~Pacing~~. The WG is also trying to make TCP more robust against packet losses, using tail loss probe (TLP) [22] or RTO restart [23]. On the other hand, there are proposals to better exploit explicit congestion notification with TCP, which is progressing as a WG item, together with formation of a new IETF WG on active queue management, a method that is needed to deploy explicit congestion notification. All current TCPM WG proposals require modifications only at TCP sender side and minor changes to the TCP stack.

Transmission control protocol Fast Open and IW10 try to cut some RTTs at the beginning of a connection in order to make TCP more responsive. A study on IW10 [20] shows that larger IW is beneficial, especially for high RTTs and bandwidth–delay product networks. The paper reports that IW10 reduces the completion time of web searches up to almost 20% compared with standard TCP when the RTT is larger than 1 s. Unfortunately, IW10 also increases the amount of packet drops and retransmissions. The effect of packet drops on the average latency is limited. This is also because a larger IW permits a larger number of ACKs that can be used to detect and repair the losses in one RTT. However, a collateral damage may be caused on connections that share the same bottleneck.

Experiments performed within our study have investigated SPDY performance using a satellite testbed and have showed that IW10 with SPDY did not lead to significant performance improvements. One of the reasons for the lower-than-expected performance could be the increase of *cwnd* during the few round-trip exchanges carrying security information before the actual data transfer starts. When the server starts sending data, the *cwnd* has already grown larger. For the typical persistent this case, IW10 reduces the total download time by at most one RTT (i.e. 600 ms), which is a modest gain for an overall completion time of 10–20 s. If multiple short connections are used (common in HTTP/1.1 and expected for some specific uses cases with SPDY), there will be significant gain.

DOI: 10.1002/sat

Tail loss probe is being specified to reduce the negative impact on latency of packet drops at the end of a traffic burst. When the last packet (or a sequence of packets) is lost at the end of the burst, TCP cannot receive ACKs triggered by packets sent after the last packet. This causes TCP to lose its self-clocking and to wait for the RTO to expire. Because the minimum RTO (specified as 1 s) can be several times the connection RTT, waiting for a timeout delays significantly the retransmission of the last packet. The solution proposed by TLP is to keep repeating the last packet every two RTTs until the RTO expires in the absence of returning ACKs. The disadvantage of TLP is that retransmissions can mask the last packet drop, making appear as if it was correctly received. This affects the TCP congestion control in that TCP does not receive the congestion signal of a packet loss. Thus, TLP defines a sophisticated algorithm to evaluate if the last packet was effectively lost. RTO restart [23] proposes a different approach to mitigate the performance issues of timeout, targeted at thin-stream TCP applications. RTO restart refines RTO calculation reducing the timeout of up to one RTT. However, it is effective only with a small *cwnd*, while tail losses can happen also with large *cwnd*. Tail losses can significantly impact performance over long delay paths (such as satellite), where the time to recover lost segments using the RTO timer can be significant.

A similar problem is encountered when any bursty TCP application encounters congestion. New CWV [21] proposes that a TCP sender can maintain the *cwnd* open across different bursts if no congestion signals are met. This method ~~can also be~~ used with persistent HTTP, as defined in SPDY.

A possible mitigation to the bursts resulting with IW10 and new CWV is to pace packet transmissions within the RTT when TCP does not have packet in flight to 'clock' its transmissions. However, this requires a cross-layer interaction to signal the network interface to send packets at regular, interval rather than in bursts. Recent work (e.g. work by Google presented at IETF-88 in November 2013) is exploring whether such pacing is practical in typical server protocol stacks. This may enable future changes to the way TCP operates, which can reduce collateral damage and hence reduce congestion-based packet loss. This is important for satellite networks, where loss recovery can incur appreciable delay.

In summary, after many years of relatively little TCP standards development, there are now a number of ongoing initiatives that are likely to significantly change the way TCP operates for HTTP. These updates are likely to be widely deployed, and satellite intermediaries (TCP PEPs) will need to operate correctly with these updates and avoid ossification of the TCP part of the protocol stack.

5. SPDY PERFORMANCE EVALUATION OVER SATELLITE

In order to evaluate HTTP/2, we considered the Google's suite of techniques called SPDY. SPDY is a predecessor of HTTP/2, which was not initially intended to provide a new HTTP specification but contained a set of design principles to guide web browsing improvement:

(i) The increasing volume of data requires increasing protocol efficiency by reducing overheads (e.g. compressing HTTP message header and body and eliminating inessential request/response delays).
(ii) In order to speed up the download process, the two end points should be able to identify a stream of objects. In other words, streams should be given identifier so to implement out-of-order delivery.
(iii) The 'distributed' nature of web contents (i.e. data stored by different content providers) together with the need of persistent connections required more flexible policies.

The effectiveness of SPDY over a satellite network was evaluated in a series of experiments conducted using both real satellite links and emulated satellite platforms. Although the tests focussed on HTTP performance over satellite, different aspects of the interaction between application layer, network layer and the satellite ~~Mac (Apple Inc.)~~ have also been explored. Q20

This section summarises the experimental platforms that were used.

5.1. Hylas-I satellite network

In England, the University of Aberdeen (UoA) and the University of Surrey (UoS) conducted experiments over the Hylas-I satellite network using Avanti terminals, based on the Hughes IPoS Q21

DOI: 10.1002/sat

specification [24]. Figure 2 shows the experimental setup. Hosts at the UoA local area network (LAN) could connect to other hosts over the same LAN through a path crossing the satellite hop provided by the Avanti terminal. F2

To test different TCP/IP stack configuration, two computers at UoA ran two different Linux kernels: Linux kernel 3.8 including the TFO and IW10 TCP modifications and Linux kernel 2.6. Both computers were used as web server running Apache. The Apache servers [25] hosted a mirror of several popular Internet web pages with HTML modified to be static.

This testbed supported a virtual private network (VPN) that could be used to encapsulate/encrypt TCP packets and thereby disable any intermediary at the satellite terminal or at the gateway. This method to inhibit PEP processing was preferred to using the terminal interface to have maximum control of the PEP.

5.2. The National Research Council satellite field trial

In Italy, the Institute of Information Science and Technologies (ISTI) and the Institute of Intelligent Systems for Automation (ISSIA), both of the National Research Council, ran a set of tests with SPDY over an Italian Internet service provider satellite network, Link Telecomunicazioni2. A schematic of the testbed is depicted in Figure 3. The tests consisted in a set of web clients using Google Chrome requesting web pages to a proxy cache. F3

A SPDY proxy/gateway running Linux implemented policies for relaying the traffic based on Node.js [26]. Web pages were cached at the gateway in order to avoid non-deterministic behaviours caused by dynamic contents and the effects of congestion over the path to the servers.

The proxy/gateway could route the traffic over an emulated and a real satellite link. The emulated satellite link introduced an artificial delay of 520 or 720 ms and constrained the bandwidth to 128 Kb/s or 1 Mb/s, respectively. Also, in order to evaluate the impact of non-congestion losses on SPDY behaviour, the emulated link introduced random packet losses. The satellite link was accessed through an evolution X3 satellite router equipped with PEPs providing HTTP acceleration and local DNS caching.

This testbed also used a PEP to perform TCP acceleration; this was used with connection splitting to the space communications protocol specification–transport protocol. To access the satellite channel, a remote terminal located within the provider's headquarter in Genoa was used; data were then routed to the hub located in Lucca.

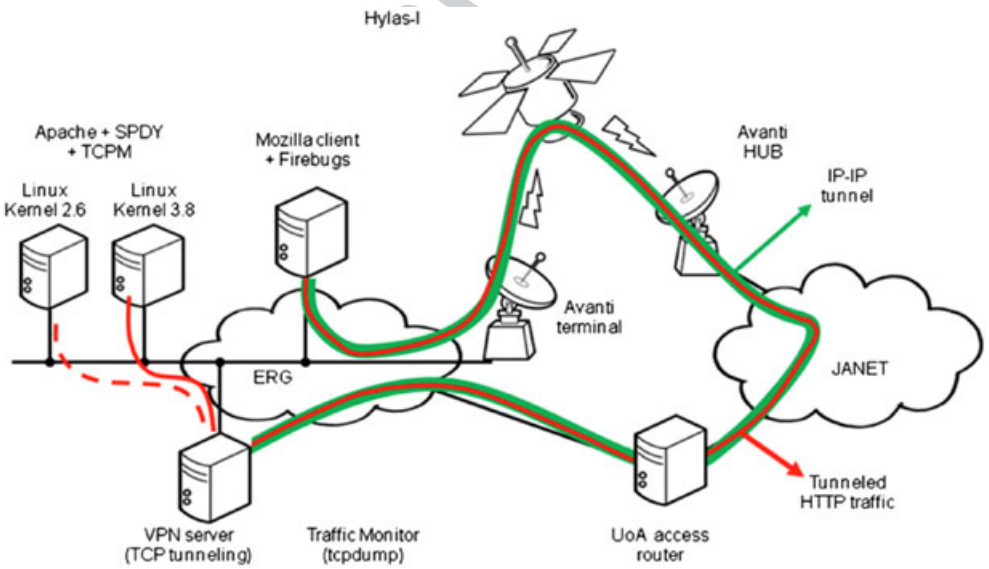


Figure 2. Hylas-I satellite testbed for SPDY evaluation. TCP, transmission control protocol; UoA, University of Aberdeen; VPN, virtual private network.

DOI: 10.1002/sat

Colour online, B&W in print

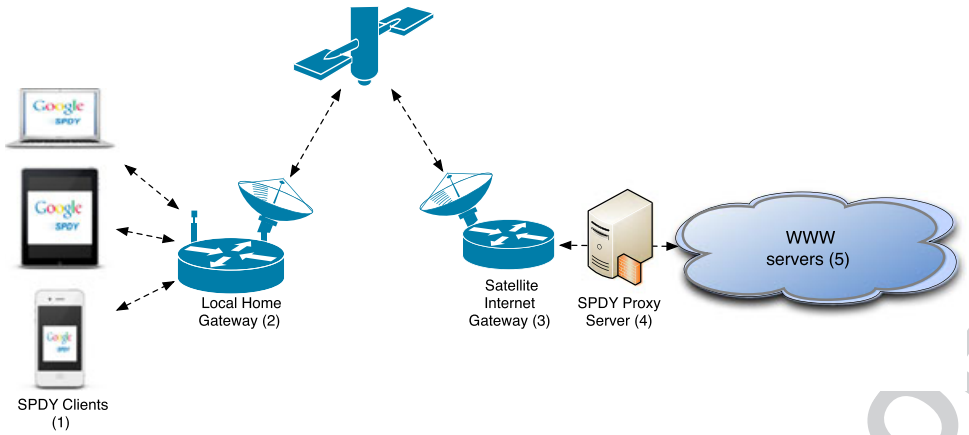


Figure 3. Institute of Information Science and Technologies/Institute of Intelligent Systems for Automation satellite field trial. WWW, World Wide Web.

5.3. Simulated satellite environment with WiFi access at the University of Bologna

The University of Bologna (UoB), in Italy, simulated the scenario of web browsing by means of wireless portable devices connected accessing the Internet through a WiFi linked to a satellite terminal. The satellite link, which plays the role of the access network, was emulated using Dummynet [27], to reproduce the typical delay and bandwidth values of a satellite link.

The testbed (Figure 4) consisted of three parts: F4

- Several web clients running different browsers on different hardware platforms (e.g. a Nexus 7' tablet, an iPad 2 tablet and a Mac OS X laptop). The mobile devices are connected through a dedicated WiFi network.
- A satellite gateway also emulating the GEO satellite link. This is emulated by inserting the typical propagation delay of GEO satellite (RTT = 600 ms) by means of the Dummynet [27] emulation tool. Q22
- A benchmarking web server, which is run by Node.js[§] (version 0.10.2). There are not intermediaries, such as Apache[¶], between the server and the client to achieve the least possible latency.

The goal of the experiments was to evaluate how the protocols and their different implementations perform in a GEO satellite scenario. In particular, this evaluated the different protocols and techniques for implementing real-time web applications (i.e. ALP, SSE and WS) in the presence of a satellite delay and to investigate testing both desktop and mobile devices.

5.4. The satellite network emulator platform at the University of Rome2

The University of Rome2 (URome2), in Italy, analysed how SPDY traffic interacted with various BoD mechanisms and evaluated the efficiency of usage of satellite link resources. URome2 tests were based on the satellite network emulation platform (SNEP), developed at the University of Rome, which emulates a satellite digital video broadcasting (DVB)-RCS network [28]. In addition, URome2 set up an Q23
Apache web server with SPDY to investigate the impact of server push. The web server was accessed through a Google Chrome web browser equipped with a page benchmark extension tool [29].

Satellite network emulation platform [28] is a satellite virtual network where the behaviour of components of a DVB-RCS network is emulated on a virtual environment managed by the WMware™ vSphere Hypervisor ESXi [30]. Each virtual node emulates a component of a VSAT network based Q24 Q25
on DVB-RCS [31]. The emulated components of SNEP are as follows: the gateway (access router), the network control centre; the satellite; the satellite terminals; and the user terminals, connected to

[§]https://nodejs.org
[¶]http://httpd.apache.org

DOI: 10.1002/sat

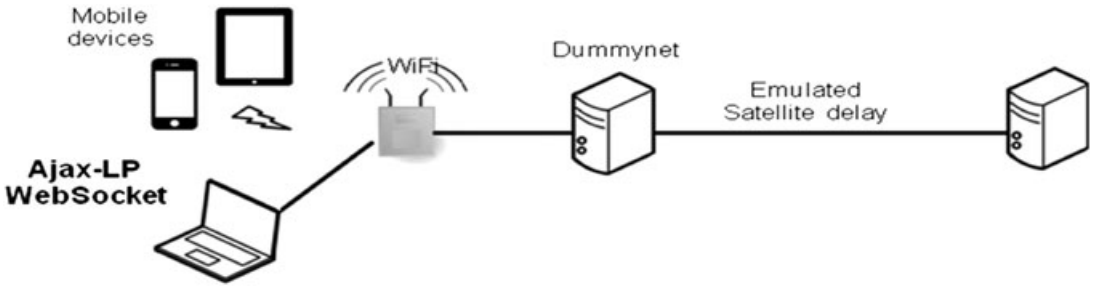


Figure 4. Emulated satellite network with WiFi access.

the satellite terminals through a LAN. Figure 5 shows the interconnections between emulated nodes in SNEP and real external nodes. Connection to external entities is required to test the interoperation of the DVB-RCS network with other systems. F5

6. PERFORMANCE COMPARISONS: HTTP/1.1 AND SPDY

6.1. Comparisons between SPDY and HTTP/1.1 using a virtual private network connection

Figure 6 compares the number of TCP connections required to download a web page with respect to the number of objects in three scenarios at UoA: via the satellite link, via the satellite network using a tunnelled connection (VPN) and by using a simulated delay of 600 ms (Netem). F6

The results show that the number of connections is reduced to 1 or 2 by using SPDY with respect to 10–20 by using HTTP/1.1. Also, the number of connections does not depend on the number of objects or the web page size, which is useful to reduce the per-connection burden at any intermediate nodes. Q26

In some cases, SDPY opened an extra connection to the server when the first connection became unresponsive. To explore this, the delay was artificially inflated by using Netem, showing that this could cause SPDY to open a new connection or fall back to HTTP/1.1.

Lowering the number of connections did not increase the latency. On the contrary, the high level of multiplexing and the reduced overhead contribute to reduce significantly the page loading time (PLT) in many cases. Figure 7 shows that, when accessing the web server through the satellite, the delay increases up to 100% with two different kernel configurations. Tests with SPDY verified that a satellite gateway was not able to operate any HTTP acceleration as SPDY concealed the HTTP header using TLS. F7

A configuration that used a VPN connection prevented TCP split connections and hence eliminated the benefit of transport PEP. The performance of SPDY and HTTP/1.1 degraded, and SPDY was able to provide only marginal improvements with respect to HTTP/1.1 (Figure 8). This may indicate that the F8

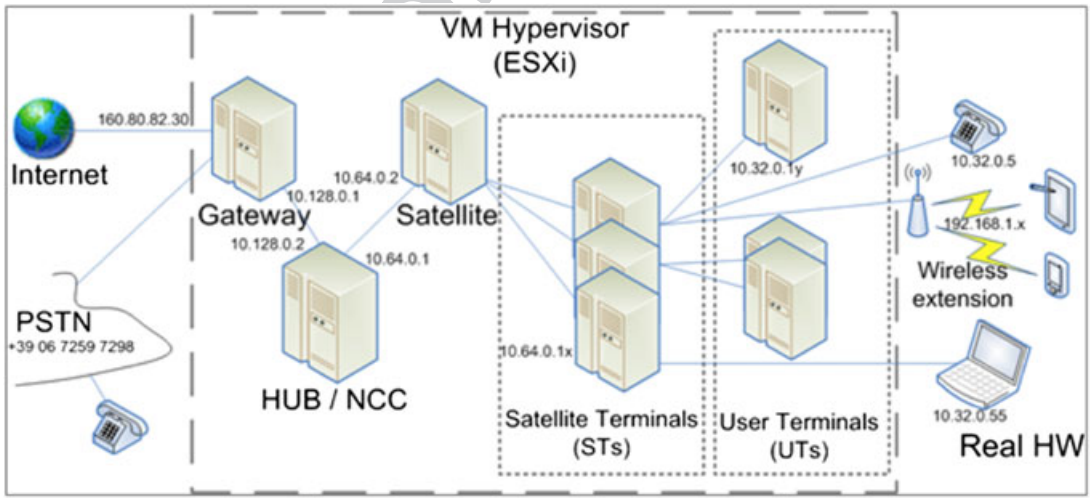


Figure 5. University of Rome2 satellite network emulation platform architecture.

Colour online, B&W in print

DOI: 10.1002/sat

Colour online, B&W in print

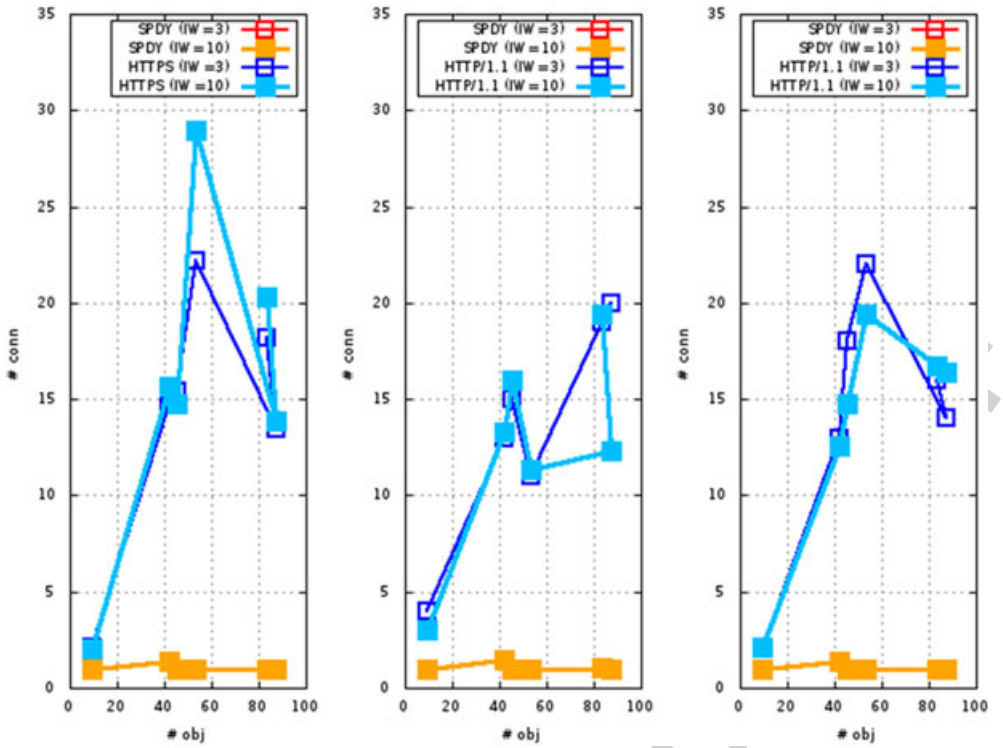


Figure 6. Number of transmission control protocol connections in HTTP/1.1 and SDPY (University of Aberdeen testbed).

Colour online, B&W in print

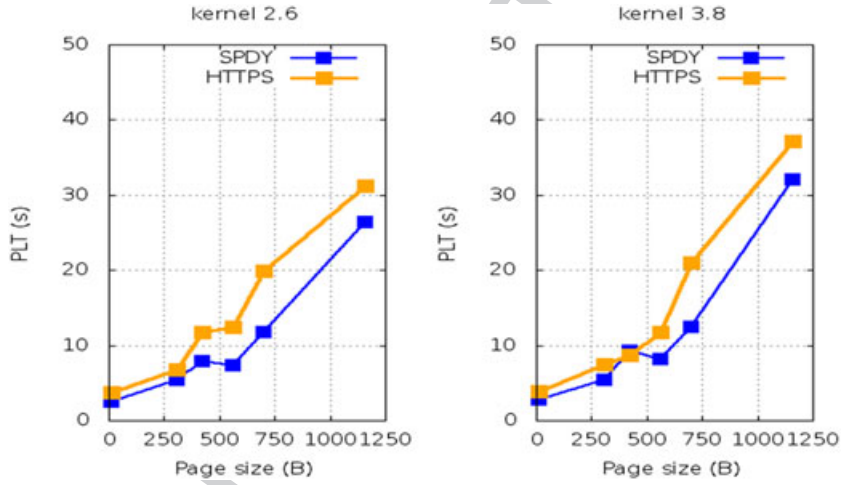


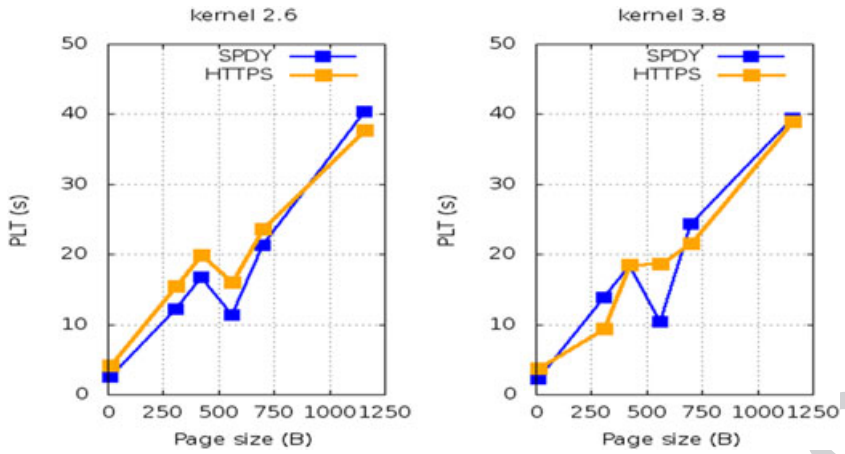
Figure 7. Comparisons between HTTP/1.1 and SPDY in Hylas network (University of Aberdeen). PLT, page loading time.

present design of SPDY is not designed for large RTTs and not yet able to effectively adapt to a large RTT. It does, however, confirm that transport PEP has benefit for SPDY.

6.2. Comparisons between SPDY and HTTP/1.1 using a proxy/gateway

SPDY and HTTP/1.1 have been compared on the ISTI/ISSIA testbed platform based on an emulated link and a real satellite network, introducing 1% random losses over the terrestrial link. The emulated satellite RTT was varied from 520 to 720 ms. These are realistic figures for a satellite link (iDirect

DOI: 10.1002/sat



Colour online, B&W in print

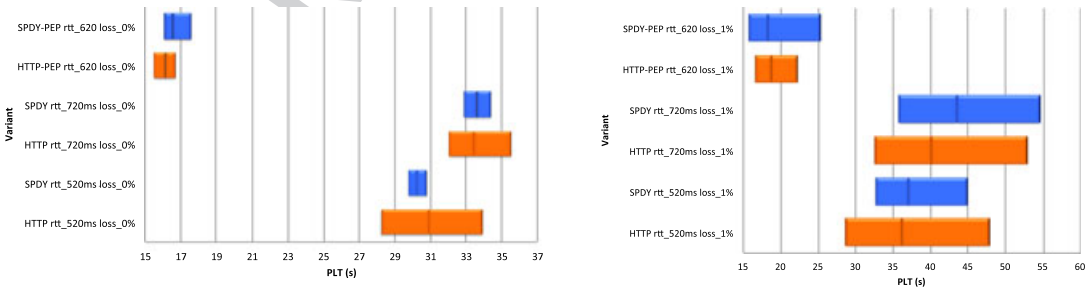
Figure 8. Comparisons between HTTP/1.1 and SPDY over Hylas-I network using a virtual private network connection to inhibit transmission control protocol split connection. PLT, page loading time.

terminal) that introduced around 640-ms delay. Figure 9 plots the 95th percentile of web page download times averaged over seven popular sites. The plot shows that a (transport) PEP can significantly reduce the web page loading both with and without errors. This result confirms the UoA results that end-to-end transport-layer optimisation is beneficial. F9

SPDY and HTTP/1.1 achieve very similar results in this case, even if SPDY outperforms in terms of the smaller interval between the minimum and the maximum download times in the lossless scenario (left picture in Figure 9). This differs from UoA–UoS analysis. The reason is likely due to the difference between fetching from a native SPDY web server and passing through a SPDY proxy. In fact, the HTTP proxy in this test when SPDY was disabled simply mapped the incoming sockets with as many correspondent outgoing ones. Instead, a SPDY proxy had a single incoming socket, from the SPDY client, but had to open many HTTP connections toward the HTTP content servers. With a native SPDY server, a single socket traverses the entire end-to-end path. Thus, passing from many parallel HTTP connections (between the HTTP web server and the SPDY proxy) to a single 'SPDY' TCP connection (between the SPDY proxy and the SPDY browser) introduces a multiplexing issue and a queuing problem that result in the differences from UoA–UoS analysis.

Figure 9 also illustrates in the right picture the performance when SPDY is used over a lossy channel. In this case, it exhibits a slightly lower performance on average than HTTP/1.1 for a range of RTTs. The reason is that SPDY uses a single connection and therefore, it is less robust to non-congestion losses than HTTP/1.1 that uses more parallel connections. Indeed, a random loss only affects one of the several parallel HTTP/1.1 transactions and is less likely to slow down the entire download process.

Figure 10 shows a breakdown of the PLT with respect to the net page size (i.e. without HTTP overhead) for the set of investigated websites. There is no direct proportional relation between the amounts of data transferred and the PLT. The PLT is influenced by other factors, such as the page structure (i.e. the web of dependence between objects), the level of optimisation of Javascript code and the distribution of data across different hosts. However, there is a trend of proportionality between F10



Colour online, B&W in print

Figure 9. Effect of packet losses on minimum, maximum and mean web page download time (Institute of Information Science and Technologies/Institute of Intelligent Systems for Automation testbed). PLT, page loading time.

DOI: 10.1002/sat

Colour online, B&W in print

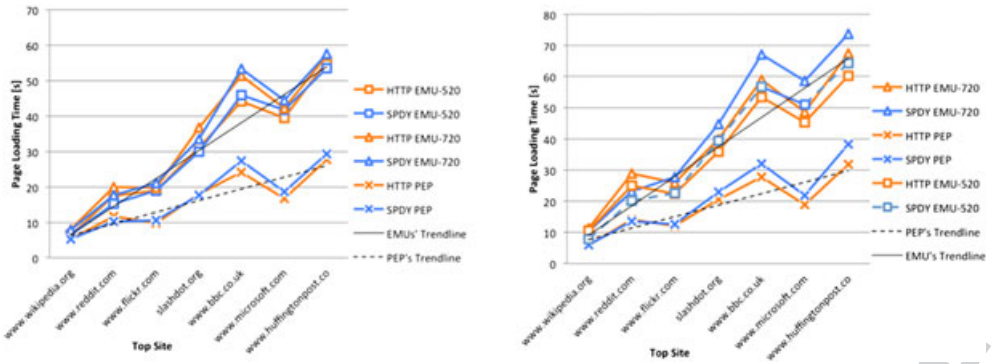


Figure 10. Comparisons of web page load time per site over (Institute of Information Science and Technologies/ Institute of Intelligent Systems for Automation) testbed in case of no added random losses (left figure) and with added losses (right figure). PEP, performance-enhancing proxy.

web page size and the PLT as spotlighted by the interpolating lines. This figure shows that major differences in performance between SPDY and HTTP/1.1 occur for larger web page sizes when there are non-congestion losses (e.g. from a wireless link forming a part of the network path). Indeed, longer data transfers would benefit more from a larger *cwnd*, but random losses induce TCP to reduce *cwnd* so that more RTTs are needed to complete the transfer.

7. EVALUATING BANDWIDTH ON DEMAND MECHANISMS WITH SPDY

Tests performed by URome2 compared the performance of SPDY and HTTP/1.1 in different configurations using SNEP. The focus was on BoD mechanisms. The transmission of the first few segments from the web client BoD was particularly critical when a bandwidth allocation was not available for a terminal. In the following set of experiments, three BoD categories of DVB-RCS: constant rate allocation (CRA), rate-based dynamic capacity and volume-based dynamic capacity (VBDC) [31].

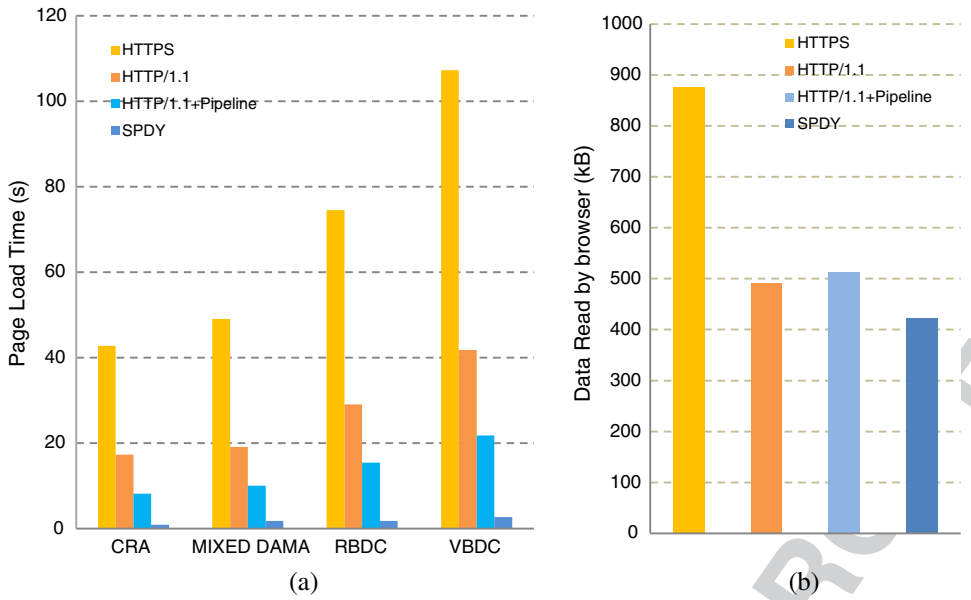
The tests used a static web page with a large number of images (640 small images) stored at the server. Tests with this page highlight the poor performance of HTTP/1.1 over high RTT links and show how SPDY can offset performance using multiplexing, header compression and server push.

Figure 11 shows test results with the test web page in terms of PLT, the number of connections and **F11** the amount of data sent in different HTTP configurations. It compares HTTPS, HTTP/1.1 with no persistent connections, HTTP/1.1 with persistent connections and SPDY. If HTTP/1.1 persistency is not used, every object requires establishing a new TCP connection resulting in huge latencies. The PLT is even larger with HTTPS where security exchanges take place with additional connection before every HTTP transaction. SPDY performs better than the other protocols in all cases, and the performance improvement increases at higher RTTs. Moreover, the best performance is achieved using a mixture of rate-based dynamic capacity and VBDC. In this test, header compression in SPDY does not have much impact compared with HTTP/1.1. However, HTTP/SSL requires more bytes (870 Kbytes) because of the encryption and security handshaking.

An important QoS parameter is the 'Time of the First Paint', that is, the delay between a request and the start of web page rendering. In general, it was observed that HTTP/1.1 was able to render the page faster than SPDY. SPDY is indeed hindered by multiplexing, header compression and framing operations. The time required by SPDY to start rendering is close to the time required by SSL, while the same time for the first paint in HTTP/1.1 pipelining is similar to the normal HTTP/1.1 (Figure 12). **F12**

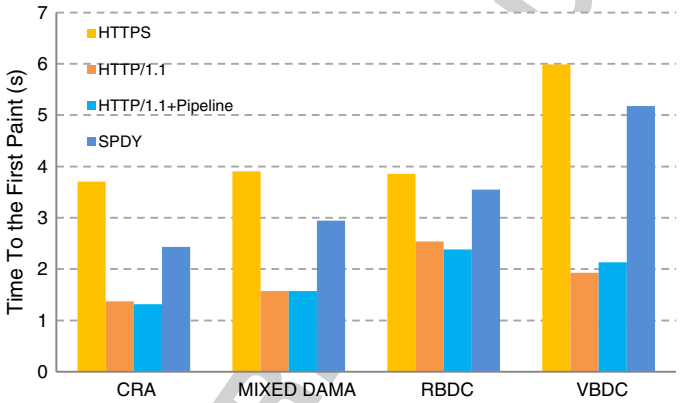
Several tests considered SPDY with different percentages of pushed objects over different BoD settings. Results show that SPDY download time is quite independent upon the percentage of pushed objects. This is a server-side configuration choice. Still, overall performance depends on the experienced RTT. As in other studies, a correctly configured CRA method was shown to outperform the other schemes since leading to an RTT close to the two-way propagation delay only; however, over-allocation using CRA can result in poor satellite resource utilisation. On the other hand, VBDC provides the worst user performance due to an overall RTT of about 1.6 s, with the highest efficiency of

DOI: 10.1002/sat



Colour online, B&W in print

Figure 11. (a) Page loading time for different bandwidth on demand methods with HTTPS, HTTP/1.1 no persistence, HTTP/1.1 persistence and SPDY. (b) Amount of data sent by each protocol. CRA, constant rate allocation; RBDC, rate-based dynamic capacity; VBDC, volume-based dynamic capacity.



Colour online, B&W in print

Figure 12. Time of the first paint for four HTTP configurations. CRA, constant rate allocation; RBDC, rate-based dynamic capacity; VBDC, volume-based dynamic capacity.

satellite resource utilisation. Other mechanisms can be used to trade off these efficiencies but are beyond the scope of the current project.

8. WEB REAL-TIME APPLICATIONS OVER SATELLITE

Tests at UoB examined three typical scenarios of real-time web applications over satellite: a server update, a client update and a chat. Server and client updates consisted of a sequence of hundreds of small HTTP messages generated to update the server and client, respectively. The chat scenario consisted in exchanging 100 messages in both directions. These scenarios were used to evaluate the different real-time web configurations across the majority of mobile and desktop web browsers. The evaluated server/client configurations for web real-time apps with satellite delay are as follows: the ALP, SSE and WS. The evaluated browsers are as follows: Google Chrome (version 27.0.1453), Firefox (version 21.0.0), Safari (version 6.0.4), iOS™ (mobile Safari version 6.0.0) and Android™ (mobile Chrome version 18.0.1025).

DOI: 10.1002/sat

The bar charts in Figure 13 compare ALP, SSE and WS with different browsers; an emulated satellite introduced a delay of 600 ms. ALP takes always at least two RTTs (i.e. about 1.2 s) to complete an update. The largest delay and delay jitter are observed for mobile devices, iOS™ and Android™ OSes, due to the WiFi access. F13

WebSocket outperforms SEE and ALP and exhibits a steadier and more uniform behaviour across different browsers. Indeed, the server update completes in one RTT with WS as opposed to two RTTs with SEE. In fact, the SEE performance was caused by a bad interaction with TCP Nagle's algorithm in the presence of large delays (TCP Nagle is often disabled for interactive services).

These results highlight the importance of a bidirectional communication channel with predictable latency, such as the one provided by WS or SPDY. This is particularly important for mobile nodes that subject to larger delay jitter induced by the wireless access.

9. IMPACT ON THE SATELLITE NETWORKS

Our experience with using SPDY over real and emulated satellite networks demonstrated that web performance can benefit from the advances offered by the new technologies, and we expect this also to be true for HTTP/2. Some pages will see more benefit with SPDY than others (e.g. the well-known flags page is known to particularly benefit from HTTP/2, a site bringing content from many origin servers or using applications layered above the web may not show benefit). We expect these results to change in favour of HTTP/2 as the protocol implementations mature, and experience increases in hosting HTTP/2 servers.

The research found that the use of dynamic bandwidth allocation did not play a major role in determining the download performance of HTTP in a satellite network. Indeed, when we disabled the PEP, it was found that the latency of the web page download was comparable with the performance of an emulated satellite delay (Netem). Because the experiments in Netem do not include BoD schemes, it was concluded that the delay usually met in satellite links (650 ms) is the major bottleneck for web page download and existing method makes sufficient capacity available on the uplink to transport HTTP requests and TCP acknowledgments. Data upload functions are likely to be more influenced by the design of request/allocation methods. The transport PEP (e.g. split TCP) benefits a SPDY session more than a HTTP/1.1 session. This was explored in experiments at UoA/UoS, and ISTI/ISSIA connected a web client to a server through a satellite network with a PEP that used TCP splitting.

Some experiments inhibited transport PEP using a tunnel (VPN experiments), which allowed an IP-encapsulated connection between the client and the server. In this case, the performance gain of SPDY was much less evident across the entire set of web page lengths. A similar result was also found in other laboratory experiments where we artificially increased the RTT between client and server. When the RTT was increased to more than 200 ms, pages using SPDY tended to open more than one connection effectively behaving as HTTP/1.1. This is evidence that SPDY is not a simple transaction protocol but is adaptive to the network path characteristics that it encounters. We concluded that current SPDY

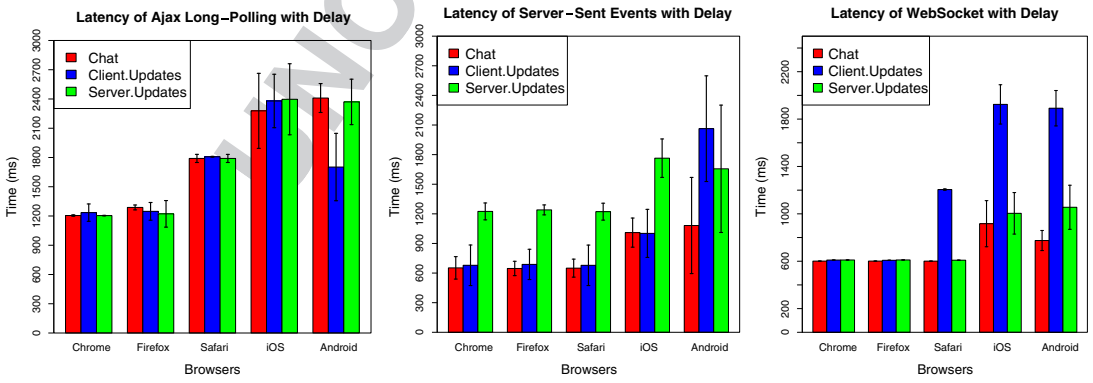


Figure 13. Comparisons of page loading time with Ajax Long Polling, server-sent events and WebSocket in three scenarios: client update, server update and chat.

DOI: 10.1002/sat

implementations are best tailored for short RTTs and that results over the satellite can result in variable performance that can sometimes be hard to explain but appears to result in different strategies for using the network (a simple example is using different numbers of TCP connections).

Methods are needed to facilitate application-layer acceleration in PEPs where this is still needed. One solution could be to define a method to reveal to the end nodes the presence of intermediaries operating on an HTTP flow and to allow the end nodes explicit interaction with intermediaries, which requires a new approach. This and the use of server push are areas in which new specifications are likely to emerge.

The work also noted the increase in the number of updates to TCP that are designed to enhance HTTP/2 performance. In considering this, we note that transport PEP can result in ossification of equipment when the PEP is not continuously updated to support new transport options and hence forces receivers to revert to using older protocol features that rely on enhancement by the PEP. Future PEP designs need to cater for incremental evolution of the protocol stack—where some sites adopt new methods and some are still to do ~~so, while some may not update their protocol stack.~~ This will become more of an issue as a wider variety of transport protocol features become used across web server platforms.

Two outcomes from evolution of TCP are possible:

- Optimistic: In the long run, it may be possible that the enhancements to standard TCP accommodate better the fundamental issues posed by satellite networks. That is, large path delays, occasional packet loss and the variability presented by RRM protocols below IP. Split TCP is important now, but this view suggests that at some future time, split TCP will no longer be needed in a PEP, which would increase transparency of the satellite network to further evolution of the transport protocols. Q27
- Pessimistic: The changes driven by introduction of HTTP/2 could result in many changes to transport behaviour that are targeted at commonly deployed web clients and which perform poorly in a satellite context. New protocols could be introduced (e.g. possibly encrypted protocols running over user datagram protocol) that may be difficult to intercept in the way that split TCP has accelerated TCP for HTTP/1.1. This will result in a need to design and deploy more intermediary functions within satellite networks and will make the design of transport-layer PEPs increasingly complex and difficult to sustain.

A key challenge to achieving the optimistic conclusion for satellite networks is to track the developments being proposed in standards groups in both transport and applications space to ensure that new changes are robust to the path delay and other characteristics presented by satellite links. Similar action is required to address the needs of cellular networks, but there are also differences in the way the networks operate that may require both communities to participate in the current standards initiatives.

10. CONCLUSIONS

At the application layer, HTTP/2 changes many things. It will be much more adaptive and network friendly than current HTTP/1.1. We expect HTTP/2 to help reducing the HTTP overhead and improving the performance for satellite. These gains come at a cost, because they impact the design and operation of satellite networks.

HTTP/1.1 content delivery often employs sharding, where web objects are purposely distributed over multiple web servers, requiring many transport connections to retrieve a single page. SPDY instead can multiplex many objects onto a single connection. Sharding is therefore much less attractive. Other decisions on content placement are likely to evolve as SPDY is more widely deployed. From a satellite perspective, fewer concurrent connections reduce the state required when network address port translation is used and eases classification of flows.

While HTTP/1.1 is often seen as a simple get/response protocol, SPDY will be flexible enough to allow various forms of client/server optimisation. With SPDY, a sender can decide not only how to multiplex web objects onto the transport but also how many parallel transport connections to use. As evidence of this tuning, we note that present versions of SPDY can lead to variable, sometimes unexpected performance over satellite networks. In addition, content placement and organisation may

DOI: 10.1002/sat

significantly impact SPDY performance. We expect HTTP/2 adaptation and tuning to evolve with time. There is a concern that the evolution of HTTP/2 could not accommodate the requirements of large delay networks (because these users represent a small proportion of the users of the general Internet). This concern motivates a recommendation for satellite system operators and equipment manufacturers to track the standards evolution and actively engage in relevant discussions, in order to ensure that the concerns of the satellite community are adequately represented.

One important area of impact concerns the way satellite PEPs are designed and used. PEPs are widely deployed in present satellite networks. They have been crucial to providing acceptable performance using HTTP/1.1. To accelerate performance, a PEP operates at multiple layers influencing the lower layers (e.g. RRM), the transport (e.g. split TCP) and applications (e.g. compression, pre-fetching and HTTP proxy). When thinking about SPDY and HTTP/2, it is important to differentiate the PEP functions into transport, application and RRM mechanisms. This suggests that the currently combined role of applications and transport PEPs could be better viewed as two separate roles, with the future possible use of standard configured proxy for web acceleration.

The use of a transport PEP has benefit for SPDY sessions, and this is recommended for current use of HTTP/2. In the future, continued evolution of the transport protocol could reduce this need. Alternate transports for SPDY have been proposed that either update TCP or do not use TCP, and it is likely that these cannot be accelerated using current PEPs. Transport support in PEPs therefore needs to track developments in the transport protocols and the way the transport is used. This is essential to ensure that satellite systems do not fall behind other network technologies.

The use of application PEP with HTTP/2 will require using an HTTP proxy. The use of proxies is different for SPDY because of its different security model. At present, intercepting application-layer accelerators for web content, such as TurboPage, are not compatible with SPDY's current use of SSL. If these are not changed, this could eliminate a useful set of optimisation tools for the satellite community. Support for configured HTTP proxies is likely to have significant benefit for cacheable content. There is work to enable proxy functions in HTTP/2 that will need to be leveraged to enable satellite PEPs to intercept and forward HTTP/2 content. Server push has many features in common with pre-fetching but also differences. Future use of server push method could be associated with the broadcast/multicast capabilities of a satellite forward link.

Many PEPs are cross-layer devices and hence can also influence the use of the lower-layer satellite service using dip packet inspection to classify flows and then match the flows to lower-layer functions. Use of alternate transports and encryption will require satellite networks to find other ways to identify traffic requirements, for example, use of differentiated services. Further research may be required to identify the implications of this change in the architecture.

In conclusion, this work urges the satellite networking community to increase awareness of the development of the new HTTP technologies and the implications that this will place on the design of the higher-layer packet processing. Furthermore, the new protocols are likely to continue to evolve and are expected to be deployed incrementally. Therefore, satellite systems need to plan for change and to understand the implications on satellite network design/configuration.

There is also a need to influence the present development of standards at both the HTTP and transport layers to ensure that new specifications do not prejudice performance over long delay paths and evolve in a way that ensures that satellite continues to offer high-quality service comparable with terrestrial networks. This has implications on the design and evolution of satellite PEP to avoid performance problems when the new protocols are deployed.

REFERENCES

1. Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P, Berners-Lee T. RFC 2616, hypertext transfer protocol – HTTP/1.1, 1999. http://www.rfc.net/ rfc2616.html.
2. WebSiteOptimization, [online article], The average web page size triples since 2008, Nov 2012.
3. Allman M, Paxson V, Blanton E. TCP congestion control, RFC 5681, IETF Standard Track, Sept 2009.
4. Belshe M, Peon R. SPDY protocol – draft 3, 2012. http://www.chromium. org/spdy/spdy-protocol/spdy-protocol-draft3.
5. Roskind J. QUIC (quick UDP Internet connections), multiplexed stream transport over UDP, Google technical report, [online], Apr 2012.
6. Iyengar J, Cheshire S, Greassley J. Minion – service model and conceptual API, Internet draft draft-iyengar-minion-concept-01, Jul 2013.

DOI: 10.1002/sat

7. Belshe M, Twist, Peon R, Thomson M, Melnikov A. Hypertext transfer protocol version 2.0, IETF, draft-ietf-httpbis-http2-09, Dec 2013. Q28
8. Dierks T, Rescorla E. The transport layer security (TLS) protocol version 1.2, RFC 5246, Standards Track, August 2008.
9. Freier A, Karlton P, Kocher P. The secure sockets layer (SSL) protocol version 3.0, RFC 6101, Historic, August 2011.
10. Border J, Kojo M, Griner J, Montenegro G, Shelby Z. Performance enhancing proxies intended to mitigate link-related degradations, RFC 3135, Informational, June 2001.
11. Radhakrishnan S, Cheng Y, Chu J, Jain A, Raghavan B. TCP Fast Open, ACM Conext 2011, Tokyo, published online, Dec 2011.
12. Ramachandran S. Web metrics: size and number of resources, http://code.google.com/speed/articles/web-metrics.html
13. TurboPage with active compression, Hughes (R), http://www.hughes.com/technologies/satellite-systems/hughes-technology/turbopage-with-activecompression, Sept 2012.
14. WalkerSands Communication. Quarterly web traffic report, http://www.walkersands.com, Chicago, Jan 2013.
15. Fette I, Melnikov A. The WebSocket protocol, RFC 6455, Internet Standard, Dec 2011.
16. Bozdag E, Mesbah A, van Deursen A. A comparison of push and pull techniques for AJAX. Web Site Evolution, 2007. WSE 2007. 9th IEEE International Workshop on, vol., no., pp.15, 22, 5-6 Oct. 2007, doi: 10.1109/WSE.2007.4380239
17. Hickson I. Server-sent events, W3C Editor's Draft, Jan 2013, http://dev.w3.org/html5/eventsource/.
18. Shuang K, Feng K. Research on server push methods in web browser based instant messaging applications, Ac. Publisher. *J Software* 2013; **8**(10):2644–2651.
19. Cheng Y, Chu J, Radhakrishnan S, Jain A. TCP Fast Open, RFC 7413, Experimental, December 2014.
20. Chu J, Dukkipati N, Cheng Y, Mathis M. Increasing TCP's initial window, RFC 6928, Experimental, April 2013.
21. Fairhurst G, Sathiseelan A, Secchi R. Updating TCP to support rate-limited traffic, Internet draft draft-ietf-tcpm-newcwv-03.txt, Oct 2013.
22. Dukkipati N, Cardwell N, Cheng Y, Mathis M. Tail loss probe (TLP): an algorithm for fast recovery of tail losses, Internet draft draft-dukkipati-tcpm-tcp-loss-probe-01.txt, Feb 2013.
23. Hurtig P, Brunstrom A, Petlund A, Welzl M. TCP and SCTP RTO restart, Internet draft draft-ietf-tcpm-rtorestart-01, Sept 2013.
24. Avanti satellite communications, http://www.avantiplc.com
25. Apache, the HTTP server project, http://httpd.apache.org/, latest release Jul 2013.
26. Indutny F. SPDY server on Node.js, https://github.com/indutny/node-spdy
27. Carbone M, Rizzo L. Dummynet revisited. *ACM SIGCOMM Comput Comm Rev* 2010; **40**(2):12–20.
28. Belli F, Luglio M, Roseti C, Zampognaro F. Evaluation of TCP performance over emulated multiple RCST DVB-RCS scenario. In Proc. Of Intl Workshop on Satellite and Space Communication (IWSSC), Siena (Italy), 424–428; 2009.
29. Chromium, benchmarking extension for Google chrome https://sites.google.com/a/chromium.org/dev/developers/design-documents/extensions/how-the-extension-system-works/chrome-benchmarking-extension
30. Wmware, Wmware vSphere hypervisor, http://www.vmware.com/ products/ vsphere-hypervisor/overview.html, US.
31. ETSI. Digital video broadcasting (DVB); interaction channel for satellite distribution systems, ETSI EN 301 790 v1.5.1, May 2009.

DOI: 10.1002/sat

Author Query Form

Journal: International Journal of Satellite Communications and Networking

Article: sat_1120

Dear Author,

During the copyediting of your paper, the following queries arose. Please respond to these by annotating your proofs with the necessary changes/additions.

- If you intend to annotate your proof electronically, please refer to the E-annotation guidelines.
- If you intend to annotate your proof by means of hard-copy mark-up, please use the standard proofing marks. If manually writing corrections on your proof and returning it by fax, do not write too close to the edge of the paper. Please remember that illegible mark-ups may delay publication.

Whether you opt for hard-copy or electronic annotation of your proofs, we recommend that you provide additional clarification of answers to queries by entering your answers on the query sheet, in addition to the text mark-up.

Query No.	Query	Remark
Q1	AUTHOR: European Space Agency. Is this the correct definition for ESA? Please change if this is incorrect.	YES
Q2	AUTHOR: Please confirm that given names (red) and surnames/family names (green) have been identified correctly.	
Q3	AUTHOR: Please provide author bio and photos for each author.	
Q4	AUTHOR: Please provide a suitable figure (abstract diagram or illustration selected from the manuscript or an additional eye-catching' figure) and a short 'GTOC' abstract (maximum 80 words) summarizing the key findings presented in the paper for Table of Content (TOC) entry.	
Q5	AUTHOR: User datagram protocol. Is this the correct definition for UDP? Please change if this is incorrect.	YES
Q6	AUTHOR: Please check if manufacturer information for this product Google™ is correct: company name, town, state (if USA), and country.	YES
Q7	AUTHOR: Working group. Is this the correct definition for WG? Please change if this is incorrect.	YES
Q8	AUTHOR: Internet protocol television. Is this the correct definition for IPTV? Please change if this is incorrect.	YES
Q9	AUTHOR: Service-level agreement. Is this the correct definition for SLA? Please change if this is incorrect.	YES
Q10	AUTHOR: Please give manufacturer information for this product TurboPage™: company name, town, state (if USA), and country.	Hughes Network Systems, Washington DC, Germantown, Maryland, USA

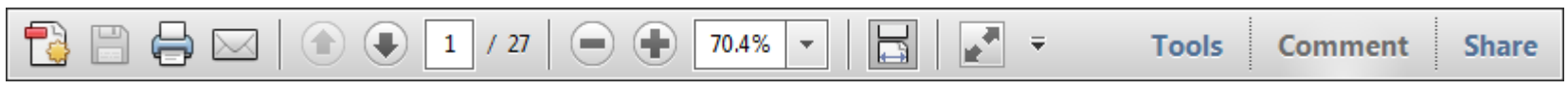
Query No.	Query	Remark
Q11	AUTHOR: Operating system. Is this the correct definition for OS? Please change if this is incorrect.	YES
Q12	AUTHOR: Please give manufacturer information for this product Android-2™: company name, town, state (if USA), and country.	Google, Mountain View, CA, USA
Q13	AUTHOR: Please check if manufacturer information for this product Apple iOS-4™ is correct: company name, town, state (if USA), and country.	
Q14	AUTHOR: Please define SEE.	Server Sent Event
Q15	AUTHOR: Please define TCPM.	TCP Maintenance and Minor Extensions
Q16	AUTHOR: Please define CWV.	Congestion Window Validation
Q17	AUTHOR: Please define RTO.	Retransmit Timeout
Q18	AUTHOR: Bandwidth–delay product. Is this the correct definition for BDP? Please change if this is incorrect.	YES
Q19	AUTHOR: Please define ACK.	Acknowledgment
Q20	AUTHOR: Please check if manufacturer information for this product satellite Mac is correct: company name.	It's acronym for "Medium Access Control"
Q21	AUTHOR: Please give manufacturer information for this product Hylas-I satellite network: company name, town, state (if USA), and country.	EADS Astrium, Paris, France
Q22	AUTHOR: Please define GEO.	Geostationary
Q23	AUTHOR: Please define RCS.	Return Channel via Satellite
Q24	AUTHOR: Please give manufacturer information for this product WMware™ vSphere Hypervisor ESXi: company name, town, state (if USA), and country.	Note: it's VMWare, VMware, Palo Alto, CA, USA
Q25	AUTHOR: Please define VSAT.	Very Small Aperture Terminal
Q26	AUTHOR: The results show that the number of.....by using HTTP/1.1. This sentence has been changed. Please check and confirm it is correct.	Correct
Q27	AUTHOR: Please define RRM.	Radio Resource Management
Q28	AUTHOR: Please provide initial for Twist.	Remove Twist (not an author)

WILEY

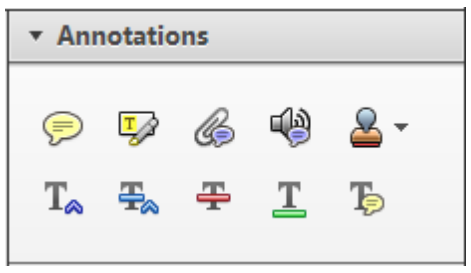
USING e-ANNOTATION TOOLS FOR ELECTRONIC PROOF CORRECTION

Required software to e-Annotate PDFs: Adobe Acrobat Professional or Adobe Reader (version 7.0 or above). (Note that this document uses screenshots from Adobe Reader X)
The latest version of Acrobat Reader can be downloaded for free at: http://get.adobe.com/uk/reader/

Once you have Acrobat Reader open on your computer, click on the Comment tab at the right of the toolbar:



This will open up a panel down the right side of the document. The majority of tools you will use for annotating your proof will be in the Annotations section, pictured opposite. We've picked out some of these tools below:

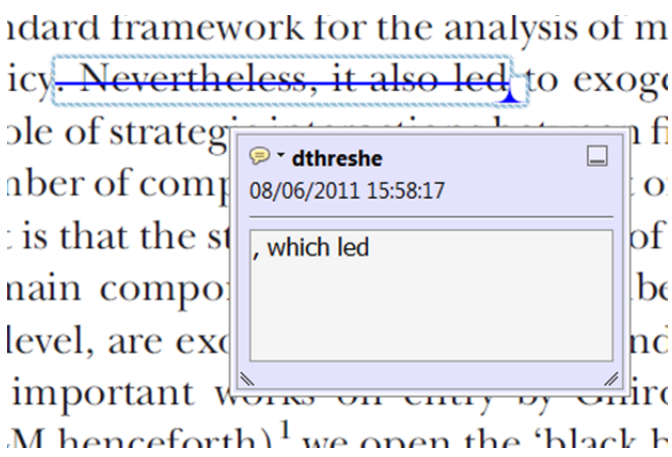


1. Replace (Ins) Tool – for replacing text.

Strikes a line through text and opens up a text box where replacement text can be entered.

How to use it

- Highlight a word or sentence.
- Click on the Replace (Ins) icon in the Annotations section.
- Type the replacement text into the blue box that appears.

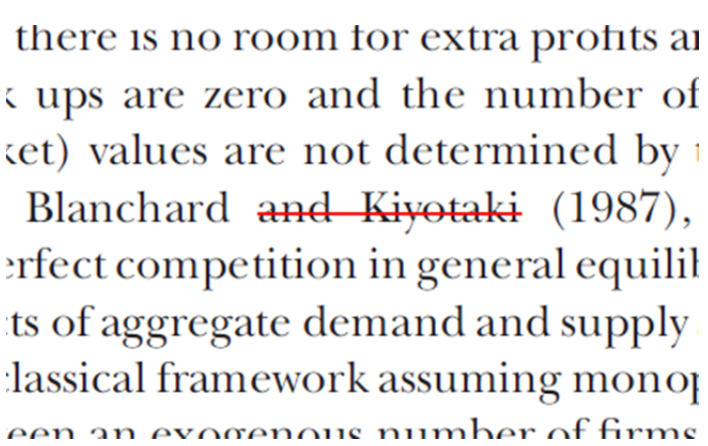


2. Strikethrough (Del) Tool – for deleting text.

Strikes a red line through text that is to be deleted.

How to use it

- Highlight a word or sentence.
- Click on the Strikethrough (Del) icon in the Annotations section.

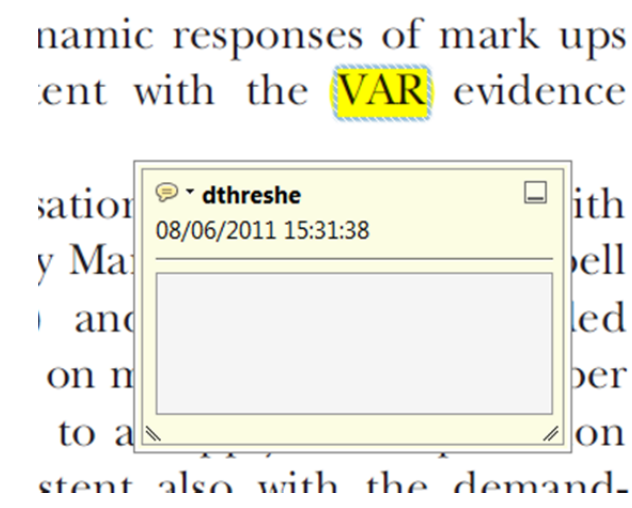


3. Add note to text Tool – for highlighting a section to be changed to bold or italic.

Highlights text in yellow and opens up a text box where comments can be entered.

How to use it

- Highlight the relevant section of text.
- Click on the Add note to text icon in the Annotations section.
- Type instruction on what should be changed regarding the text into the yellow box that appears.

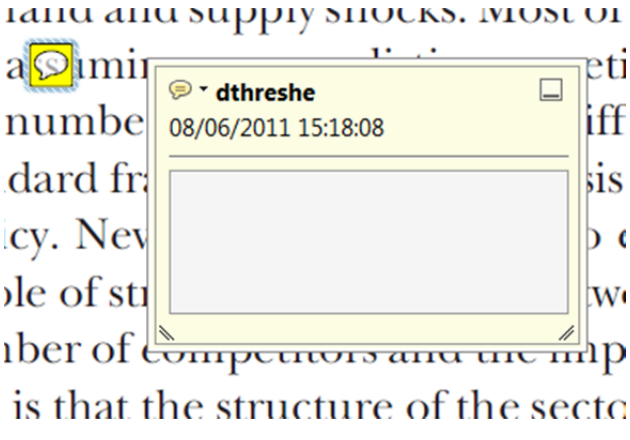


4. Add sticky note Tool – for making notes at specific points in the text.

Marks a point in the proof where a comment needs to be highlighted.

How to use it

- Click on the Add sticky note icon in the Annotations section.
- Click at the point in the proof where the comment should be inserted.
- Type the comment into the yellow box that appears.



WILEY

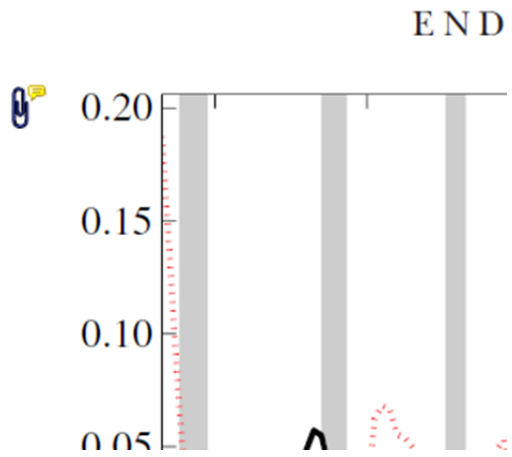
USING e-ANNOTATION TOOLS FOR ELECTRONIC PROOF CORRECTION

5. Attach File Tool – for inserting large amounts of text or replacement figures.

Inserts an icon linking to the attached file in the appropriate pace in the text.

How to use it

- Click on the Attach File icon in the Annotations section.
- Click on the proof to where you'd like the attached file to be linked.
- Select the file to be attached from your computer or network.
- Select the colour and type of icon that will appear in the proof. Click OK.

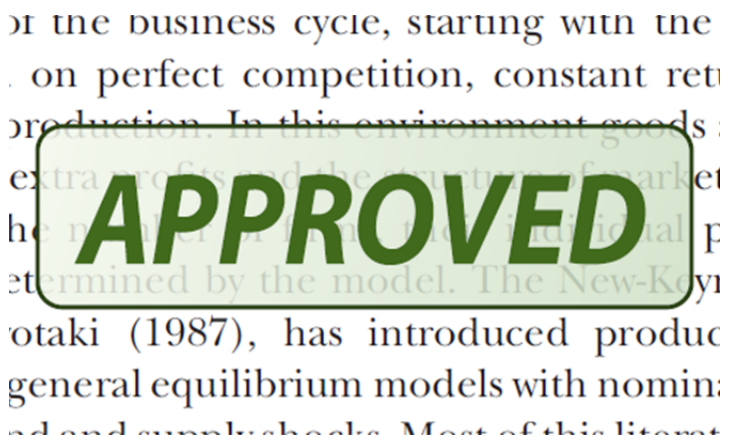


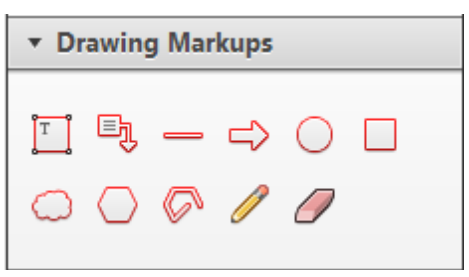
6. Add stamp Tool – for approving a proof if no corrections are required.

Inserts a selected stamp onto an appropriate place in the proof.

How to use it

- Click on the Add stamp icon in the Annotations section.
- Select the stamp you want to use. (The Approved stamp is usually available directly in the menu that appears).
- Click on the proof where you'd like the stamp to appear. (Where a proof is to be approved as it is, this would normally be on the first page).



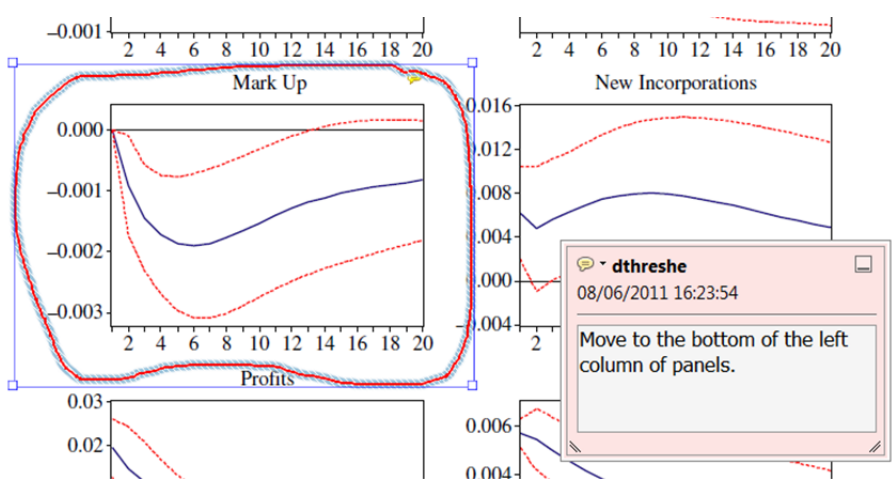


7. Drawing Markups Tools – for drawing shapes, lines and freeform annotations on proofs and commenting on these marks.

Allows shapes, lines and freeform annotations to be drawn on proofs and for comment to be made on these marks..

How to use it

- Click on one of the shapes in the Drawing Markups section.
- Click on the proof at the relevant point and draw the selected shape with the cursor.
- To add a comment to the drawn shape, move the cursor over the shape until an arrowhead appears.
- Double click on the shape and type any text in the red box that appears.



For further information on how to annotate proofs, click on the Help menu to reveal a list of further options: