Performance Evaluation of HTTP and SPDY Over a DVB-RCS Satellite Link with Different BoD Schemes

Luca Caviglione[1(✉)], Alberto Gotta[2], A. Abdel Salam[3], Michele Luglio[3], Cesare Roseti[3], and F. Zampognaro[3]

[1] Institute of Intelligent Systems for Automation (ISSIA), National Research Council of Italy (CNR), Genoa, Italy
luca.caviglione@ge.issia.cnr.it
[2] Information Science and Technology Institute (ISTI), National Research Council of Italy (CNR), Pisa, Italy
alberto.gotta@isti.cnr.it
[3] Department of Electronics Engineering, University of Rome "Tor Vergata", Rome, Italy
{abdel.salam,roseti,zampognaro}@ing.uniroma2.it, luglio@uniroma2.it

Abstract. The rapid evolution of the Web imposes the need of enhancing the HTTP over satellite channels. To this aim, SPDY is a protocol engineered to reduce download times of content rich pages, as well as for managing links characterized by large Round Trip Times (RTTs) and high packet losses. With such features, it could be an efficient solution to cope with performance degradations of HTTP over satellite. In this perspective, this paper compares the behaviors of HTTP and SPDY over a DVB-RCS satellite link. To conduct a thorough set of tests over a realistic scenario, we used the Satellite Network Emulation Platform (SNEP). In addition, we evaluated how different Bandwidth on Demand (BoD) methods impact over the retrieval of a page. Results clearly indicate that SPDY could be an effective solution to deliver Web contents over satellites in a more efficient manner.

Keywords: Networking protocol · Satellite network · BoD · SPDY · HTTP · DVB-RCS

1 Introduction

Nowadays, satellite communication is one of the preferred solutions for accessing the Internet while moving, and it is also the main choice to deploy connectivity in rural areas, or in developing Countries. Due to the physical characteristics of the link, especially long delays and high error rates, many protocols could experience performance degradations. For instance, mitigation of the impact of the high Round Trip Time (RTT) affecting GEO channels on TCP performance has been a prime research topic for years (see, e.g., [1] and references therein).

I. Bisio (Ed.): PSATS 2014, LNICST 148, pp. 34–44, 2016.
DOI: 10.1007/978-3-319-47081-8_4

However, to effectively pursue the vision of the future Internet, satellites must also handle Web traffic, which is increasing both in terms of volumes and complexity [2]. In fact, modern web pages do not primarily rely on the *main* object containing the HTML code, but they also need several *in-line* objects. The evolution of the Web heavily requires in-line objects embedding interactive services, and content-rich graphic elements. As a consequence, the legacy *page-by-page* model should be updated, along with related protocols, such as the HTTP.

To partially fulfill such issues impairing the original HTTP, the HTTP/1.1 [3] introduced multiple connections to increase concurrency, and pipelining to send multiple object requests over a single TCP connection without waiting for a response. Even if such additions improve the performance over satellites, they are not definitive [4]. In fact, the server must generate responses ordered as the requests were received, thus limiting gains and possibly leading to blocking. Nevertheless, parallelism of HTTP/1.1 is usually limited (i.e., 7 connections in standard browsers), and not supported by-default by many servers.

On the contrary, SPDY is engineered to reduce download times of content-rich pages, as well as for managing wireless channels, which can be characterized by large RTTs and high packet losses [5]. Especially, to overcome to HTTP limitations, SPDY introduces:

- *multiplexed requests* - the number of concurrent requests that can be sent over a single connection is unbounded and properly handled by a framing layer;
- *prioritization* - retrievals of in-line objects composing a page can be properly scheduled, as to avoid congestion or to enhance the Quality of Experience (QoE). For instance, the client could fetch contents enabling to "understand" a page, even if incomplete;
- *header compression* - since the more sophisticated pages may need up to 100 requests, enforcing compression prevents bandwidth wastes due to duplicated headers;
- *server push* - contents can be pushed from servers to client without additional requests.

From an architectural point of view, the previous features are grouped within an high-level framing layer, which tunnels data into a single SSL/TCP connection. Hence, SPDY could be a suitable solution for the delivery of Web contents over satellite links. While the performance of HTTP has been extensively studied in literature both for wired [6] and satellite networks [7], a complete understanding of SPDY is still an open research problem. Moreover, many works focus on evaluating the protocol over wired and IEEE 802.11/cellular mobile scenarios [8]. For what concerns satellites, from our best knowledge, [9,10] are the only previous attempts.

Therefore, this paper compares HTTP and SPDY when used over a satellite link. To this aim, we used the Satellite Network Emulation Platform (SNEP) to conduct tests on a realistic DVB-RCS environment, with different Bandwidth on Demand (BoD) schemes. The contributions of this work are: *(i)* to understand the most relevant behaviors of SPDY when used over a realistic DVB-RCS channel; *(ii)* to provide a comparison between HTTP and SPDY emphasizing the

impact of inline objects; *(iii)* to understand whether SPDY could be a suitable tool to enhance satellite communications in place of middleboxes.

The remainder of the paper is structured as follows: Sect. 2 describes the used testbed, while Sect. 3 discusses the measurement methodology. Section 4 presents the collected results, and finally, Sect. 5 concludes the paper.

2 Description of the Tesbed

To evaluate the performance of SPDY and HTTP over a satellite link we used the Satellite Network Emulation Platform (SNEP), developed by the University of Rome "Tor Vergata" [11]. Specifically, it can emulate different aspects of a DVB-RCS satellite access, for instance, the delay, the bitrate and TDMA framing, as well as BoD algorithms to dynamically assign the capacity on the return link. Figure 1 depicts the testbed implemented via the SNEP framework. Specifically, it is composed by:

- a gateway (GW) (or the access router);
- the network control centre (NCC) (or the hub);
- the satellite channel (SAT);
- the satellite terminals (ST);
- the user terminals (UTs), connected to the STs through a Local Area Network (LAN).

Each component used for the emulation is built via a Linux-based machine (version 2.6), and the needed functionalities are implemented through software modules running both in user and kernel space. To configure its behaviors (e.g., assign a fixed amount of bandwidth) a set of additional commands are made available through the Linux traffic controller `tc`. To manipulate traffic, ethernet frames are brought in the user space and then processed by an application-layer agent. The BoD portion is based on the DVB-RCS signaling, which is used to negotiate resources among different STs. Moreover, to emulate the DiffServ queuing discipline of DVB-RCS, packets are stored in a buffer implementing multiple parallel queues, which are served with different priorities.

For what concerns the hardware, both the Web client and the server are based on quad-core PCs with 32 GB RAM. To have a proper support for the SPDY protocol, we used Mozilla Firefox (24.0).

To capture data, we used the Wireshark sniffer with the SPDYShark extension enabling to decode protocol messages and to inspect its relevant parameters. When TLS/SSL encryption is used, we configured Wireshark to use the proper SSL keys to decrypt/decode the gathered data.

Table 1 reports the key configuration parameters characterizing our tests.

To emphasize the most critical behaviors when retrieving pages with different protocols, we created ad-hoc HTML pages for testing purposes. Specifically, to stress the iterations of the request-response exchange, each page contains a very large number of in-line objects, i.e., 640 small images. The main object implementing the hypertext (the `main.html`) has a size of 24.8 Kbytes. We point out that the test page has been crafted to highlight performance aspects of HTTP/SPDY when acting over high RTT links.

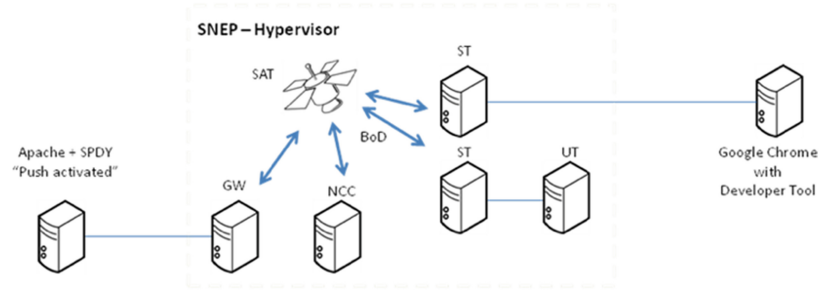


Fig. 1. Testbed used for evaluating SPDY and HTTP over a satellite network.

Table 1. TCP and Web server configurations.

TCP configuration	TCP Cubic (with optimized parameters for satellite [12])
	`tcp_moderate_rcvbuf` enabled
	`IW` = 10
Web server	Apache/2.2.22
	SPDY: Mod_SPDY 0.9.3.3
	Server Push (`X-Associated-Content header`)
	with different % for the pushed objects
	Apache KeepAlive settings enabled/disabled

3 Measurement Methodology

Planned measurements aim at comparing the behavior of SPDY with the most popular versions of the HTTP protocol, especially when different BoD mechanisms are deployed in the DVB-RCS return link. To this aim, a single user terminal (UT1) is connected to the virtual satellite terminal (ST1), which is implemented by SNEP. The available bandwidth is 4 Mbit/s and 1 Mbit/s, for the forward and the return link, respectively. The round-trip propagation delay is set equal to 520 ms, which is typical for a GEO satellite link. Besides, SNEP introduces an additional delay taking into account the MAC layer, i.e., the TDMA overhead and latencies due to the framing of the DVB-RCS. Tested BoD methods are:

- CRA (Constant Rate Assignment) - all the time slots are permanently assigned to the target station for the whole simulation duration;
- RBDC (Rate Based Dynamic Capacity) or VBDC (Volume Based Dynamic Capacity) - slots are dynamically assigned to the return link, depending on the traffic incoming to the satellite terminal: RBDC considers the ingress data rate, while VBDC uses the cumulative volume of queued data to compute capacity requests;
- mixed: a mix of the previous.

Table 2. Parameters used for emulating the different BoD schemes.

BoD scheme	Values (avg. on 50 repeated trials)
CRA	228 slots at a constant rate
RBDC	228 rate-based slots
VBDC	228 volume-based slots
Mixed	18 CRA slots, 105 RBDC slots, 105 VBDC slots

The values used for each BoD scheme are summarized in Table 2.

To have a fair assessment, we modified some parameters ruling the behavior of the browser, both for the case of HTTP and SPDY. In fact, default values are optimized for the wired Internet. In more details:

- **network.http.connection-retry-timeout:** defines the amount of time before considering a connection attempt aborted. Upon expired, the browser will open a parallel backup connection. Since this parameters is set to 250 ms by default, having an RTT of more than 520 ms would lead to unneeded TCP connections. Thus, it has been set to 0 in our trials (i.e., deactivated);
- **network.http.pipelining:** enables HTTP pipelining, i.e., the browser can send multiple `GET` without waiting for a server response. Pipelining has been enabled and set to 32 simultaneous requests, at the maximum;
- **network.http.speculative-parallel-limit:** normally, set to 6, it specifies the number of half-opened sockets to be kept for frequently used destinations (e.g., Google APIs). To avoid unpredictable behaviors of the browser, as well as to reduce overheads on the satellite link, we imposed this parameter to 0;
- **network.http.spdy.timeout:** defines the amount of time to wait after the page is considered received completely and the used TCP connection is closed (i.e., the `RST/FIN`). The default value is 180 s and is generally suitable to handle AJAX-based interactive contents. However, since our tests are aimed to verify performance during the page load, this value has been reduced to 5 s, as to not add noise to the measured times. We point out that this value is equal to the default keep-alive option of the Apache 2.2.2 used in our tests, as to guarantee a fair scenario.

The Web server has been configured to support SPDY and different versions of HTTP, i.e., HTTP1.0, HTTP1.1, SSL/HTTP1.0, and SSL/HTTP1.1. We underline that since SPDY uses SSL by default, this choice has been made to provide a more fair comparison. Trials have been performed with an instrumented Firefox browser, and each trial has been repeated 50 times.

4 Performance Evaluation

This section presents the outcome of the performance evaluation of the different version of HTTP and SPDY using an emulated DVB-RCS satellite access.

For the sake of conciseness, we present some results only using the BoD method defined as "mixed" (see Sect. 3). Nevertheless, in Sect. 4.4 we will offer a vis-à-vis comparison among the different BoD schemes.

Since it will be largely used in the rest of this section, we define the Page Loading Time (PLT) as [13]: $PLT = T_C^i - T_R$, where T_C^i is the time at which the last i-th inline object composing the page is completely received ($i = 642$ in our tests), and T_R is the time when the first `GET` for the `index.html` is performed.

4.1 Impact of the Header Compression

Native header compression is one of the most important design choice of SPDY, since it allows reducing the amount of transferred bytes. Figure 2 summarizes the amount of data transferred to retrieve the test page.

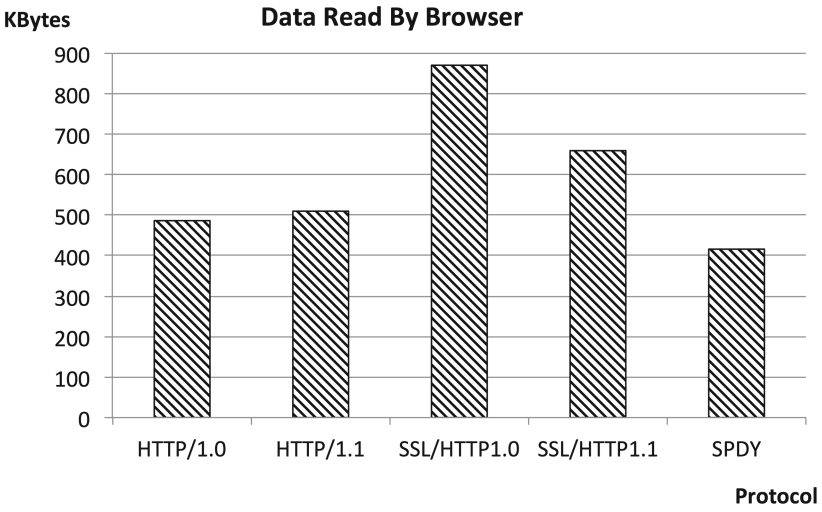


Fig. 2. Impact of the header compression per protocol.

Even if SPDY only needs 416 Kbytes to complete the transfer, such a result is very close to the cases when HTTP is adopted (∼508 Kbytes). On the contrary, the real issue preventing the effectiveness of compression is due to SSL encryption, which accounts for overheads needed for the additional handshaking. In fact, the usage of SSL with the "traditional" HTTP leads to a significant increase of the transferred data: ∼871 Kbytes for SSL/HTTP1.0 and ∼659 Kbytes for SSL/HTTP1.1. Thus, the optimized design of SPDY in managing encryption [14] definitely plays a role.

4.2 TCP Connection Analysis

Understanding how TCP connections are managed by different protocols is mandatory to enhance their behaviors over satellite. Hence, Fig. 3 compares different evolution of the transport layer against the time.

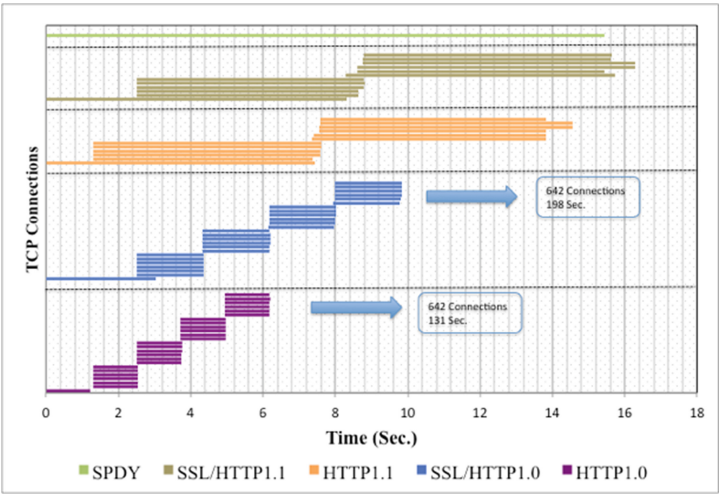


Fig. 3. Different management schemes of TCP connections per protocol.

For the case of HTTP1.0, which does not allow connection reuse, the browser opens the first TCP flow to send a `GET` for the `index.html`. As soon as it is received and parsed, a batch of 6 parallel connections is spawned to retrieve the needed in-line objects. The process is then iterated until the completion of the whole page. This leads to a PLT of 131 s (out of the graph scale), which is not acceptable. A similar evolution happens for the case of SSL/HTTP1.0 (again out of the graph scale). Yet, the need of performing additional exchanges for the SSL signaling, makes the first connection longer. Besides, encryption overheads account for longer transfer times, thus resulting into a PLT of 198 s.

When using HTTP/1.1, the first steps are still the same of the previous cases. Specifically, it uses a connection to retrieve the main object, and then uses the maximum allotted of 6 parallel TCP connections to retrieve additional contents. Then, it exploits the feature of connection reuse, and each one remains open to download 100 object (which is a limit imposed by the Apache web server). Recalling that our test page is composed by 642 objects, after hitting the limit of 600 items (i.e., 100 objects $\times$ 6 connections), a new batch of TCP connections is created. Such connections are mostly underutilized, since they are used to retrieve only a small fraction of data (equivalent to 42 objects only) (see Sect. 4.3). However, compared to HTTP1.0, the overall performance achieved in terms of PLT is better of an order of magnitude, i.e., PLT$\sim$10 s. The additional

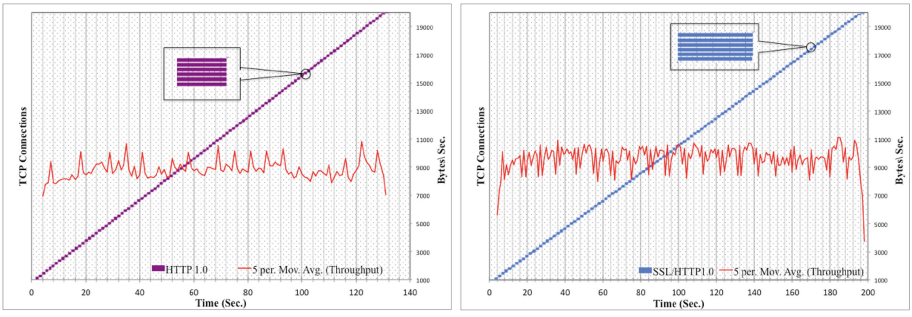
5 s, for which the connections remain active are due to timeouts of Apache (i.e., the parameter **network.http.spdy.timeout**, as discussed in Sect. 3). Also SSL/HTTP1.1 behaves similarly, with times inflated by the overheads due to SSL (as explained earlier), accounting for an additional time of ∼2 s in the PLT compared to the plain HTTP1.1.

Finally, in the SPDY case, the single-connection setup is clearly depicted. Thus, all objects are multiplexed into a unique TCP conversation. Once the page is completely received, the TCP connection is kept open by the browser for 5 s, as to maintain the same timeout period for an easier comparison. Its reduced overheads, and utilization of a unique (longer) connection, enables to better exploit the available bandwidth (even without using parallelization/pipelining). Then, SPDY has a PLT of 9 s, which is similar to HTTP1.1, but with a simpler complexity in the transport layer and supporting security. This is a plus, since satellite links are usually accessed through Performance Enhancement Proxies (PEPs) or middleboxes.

4.3 Throughput Analysis

Another important aspect to understand how the different Web protocols behave over a DVB-RCS link concerns the analysis of the throughput.

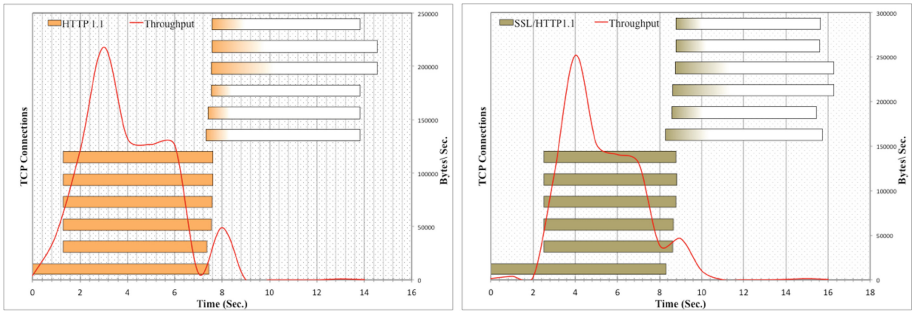
Figure 4 focuses on the HTTP1.0. Results indicate that the average rate is ∼9 Kbyte/s. A deeper analysis reveals that the main cause is due to the usage of separate connections (one per object, 642 on the overall). Therefore, for each connection, a set-up and tear-down have to be performed, also worsened by the high values of the RTT, and impairments due to the slow-start. Similar considerations can be made when SSL is used.



(a) HTTP1.0 (b) SSL/HTTP1.0

Fig. 4. Throughput analysis of HTTP1.0.

Figure 5 considers HTTP1.1. Since it implements pipelining and connection reusing, the latency impacts less on the behavior of the TCP. As a result, the achieved throughput is more than 200 Kbyte/s. Also in this case, SSL accounts for an overhead, slightly reducing the overall performances.



(a) HTTP1.1 (b) SSL/HTTP1.1

Fig. 5. Throughput analysis of HTTP1.1.

Finally, Fig. 6 shows the evolution of SPDY. Since, it uses a single connection, the delays introduced by the satellite network are absorbed (with the acceptation that they are experienced once, and not on a per-flow basis). Therefore, the achieved throughput is ∼250 Kbyte/s, which is the best obtained value in our tests.

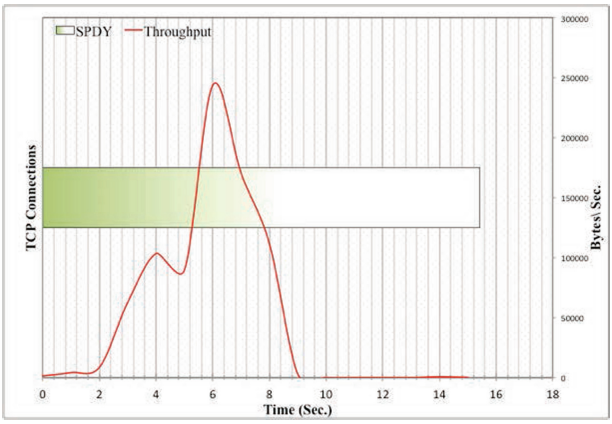


Fig. 6. Throughput analysis of SPDY.

4.4 Impact of the BoD Scheme

Figure 7 shows how the different BoD schemes impact on the PLT, for each protocol. Since previous results clearly show degradations due to the joint use of SSL and HTTP, the evaluation of the BoD scheme only considers the plain HTTP/HTTP1.1 and SPDY. In essence, the BoD increases the latency experienced by the application, which worsen the PLT. To highlight its impact, data transfers are performed on the return link.

As showcased, SPDY always outperforms the HTTP, and improvements increase for higher values of the RTTs, which characterize the VBDC and the RBDC schemes. In particular, SPDY is resilient enough to the increased latencies. In the worst case (i.e., the VDBC with an RTT of 1.6 s), its PLT is ∼21 s, that is only 10 s greater than when using CRA. On the contrary, all the flavors of HTTP are greatly impaired by the VBDC, with a PLT ranging from ∼150 s (for HTTP1.1) to ∼300 s (for HTTP1.0).

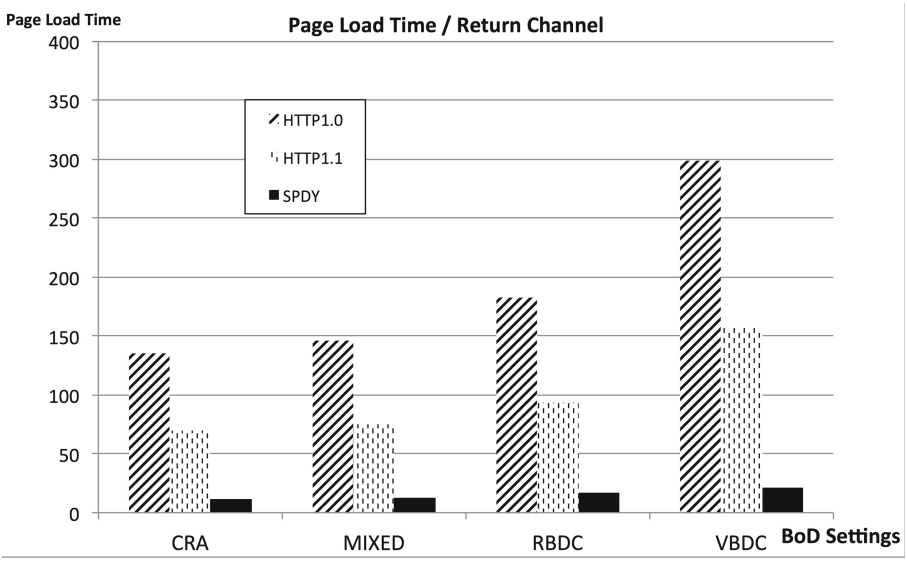


Fig. 7. PLT vs different BoD schemes.

5 Conclusions and Future Works

In this paper we compared the behaviors of different flavors of HTTP and SPDY over a DVB-RCS satellite link. To this aim, we used the Satellite Network Emulation Platform (SNEP) to conduct a thorough set of tests.

Results indicate that, owing to its single-connection architecture, SPDY is a promising solution to access the Web when using satellites. Also, it has a reduced TCP footprints, which can offload PEPs and middleboxes usually deployed in satellite service providers.

Future works aim at enriching the investigation, and also using real Internet Service Providers (ISPs) to better evaluate the feasibility of using SPDY as the unique technological enabler to bring modern Web contents via satellite platforms.

Acknowledgments. This work has been partially funded by the European Space Agency (ESA) within the framework of the Satellite Network of Experts (SatNex-III), CoO3, Task3, ESA Contract no. 23089/10/NL/CLP.

References

1. Sooriyabandara, M., Fairhurst, G.: Dynamics of TCP over BoD satellite networks. Int. J. Satell. Commun. Network. **21**(4–5), 427–449 (2003)
2. Caviglione, L.: Can satellites face trends? The case of Web 2.0. In: International Workshop on Satellite and Space Communications (IWSSC 2009), pp. 446–450, September 2009
3. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners- Lee, T.: RFC 2616, Hypertext Transfer Protocol HTTP/1.1, Network Working Group, IETF (1999)
4. Nielsen, H.F., Gettys, J., Baird-Smith, A., Prudhommeaux, E., Lie, H.W., Lilley, C.: Network performance effects of HTTP/1.1, CSS1, and PNG. ACM SIGCOMM Comput. Commun. Rev. **27**, 155–166 (1997)
5. Belshe, M., Peon, R.: SPDY protocol - draft 3, Network Working Group, IETF (2012)
6. Heidemann, J., Obraczka, K., Touch, J.: Modeling the performance of HTTP over several transport protocols. IEEE/ACM Trans. Netw. **5**(5), 616–630 (1997)
7. Kruse, H., Allman, M., Griner, J., Tran, D.: Experimentation and modelling of HTTP over satellite channels. Int. J. Satell. Commun. **19**(1), 51–68 (2001)
8. Kim, H.J., Yi, G.S., Lim, H.N., Lee, J.C., Bae, B.S., Lee, S.W.: Performance analysis of SPDY protocol in wired and mobile networks. In: Jeong, Y.-S., Park, Y.-H., Hsu, C.-H.R., Park, J.J.J.H. (eds.) Ubiquitous Information Technologies and Applications. LNEE, vol. 280, pp. 199–206. Springer, Heidelberg (2014). doi:10.1007/978-3-642-41671-2_26
9. Cardaci, A., Caviglione, L., Gotta, A., Tonellotto, N.: Performance evaluation of SPDY over high latency satellite channels. In: Dhaou, R., Beylot, A.-L., Montpetit, M.-J., Lucani, D., Mucchi, L. (eds.) PSATS 2013. LNICST, vol. 123, pp. 123–134. Springer, Heidelberg (2013). doi:10.1007/978-3-319-02762-3_11
10. Cardaci, A., Celandroni, N., Ferro, E., Gotta, A., Davoli, F., Caviglione, L.: SPDY - a new paradigm in web technologies: performance evaluation on a satellite link. In: Proceedings of the 19th Ka and Broadband Communications Navigation and Earth Observation Conference, Florence, Italy, FGM Events LLC 239, October 2013
11. Luglio, M., Roseti, C., Zampognaro, F., Belli, F.: An emulation platform for IP-based satellite networks. In: IET Conference Proceedings, pp. 712–716 (2009)
12. Ha, S., Injong, R., Xu, L.: CUBIC: a new TCP-friendly high-speed TCP variant. ACM SIGOPS Oper. Syst. Rev. **42**(5), 64–74 (2008)
13. Wang, X.S., Balasubramanian, A., Krishnamurthy, A., Wetherall, D.: Demystifying page load performance with wprof. In: Feamster, N., Mogul, J. (eds.) Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation (NSDI 2013), pp. 473–486. USENIX Association, Berkeley (2013)
14. Langley, A.: Transport Layer Security (TLS) Next Protocol Negotiation Extension, draft-agl-tls-nextprotoneg-00, Network Working Group, IETF, January 2010