Performance Evaluation of SPDY over High Latency Satellite Channels

Andrea Cardaci[2], Luca Caviglione[1], Alberto Gotta[2], and Nicola Tonellotto[2]

[1] Institute of Intelligent Systems for Automation (ISSIA)
National Research Council of Italy, Via de Marini 6, 16149, Genova
luca.caviglione@ge.issia.cnr.it
[2] Information Science and Technologies Institute (ISTI)
National Research Council of Italy, Via G. Moruzzi 1, 56124, Pisa
cyrus.and@gmail.com, {alberto.gotta,nicola.tonellotto}@isti.cnr.it

Abstract. Originally developed by Google, SPDY is an open protocol for reducing download times of content rich pages, as well as for managing channels 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 Web 2.0 services used over satellite networks. In this perspective, this paper evaluates the SPDY protocol over a wireless access also exploiting a satellite link. To this aim, we implemented an experimental set-up, composed of an SPDY proxy, a wireless link emulator, and an instrumented Web browser. Results confirm that SPDY can enhance the performances in terms of throughput, and reduce the traffic fragmentation. Moreover, owing to its connection multiplexing architecture, it can also mitigate the transport layer complexity, which is critical when in presence of middleboxes deployed to isolate satellite trunks.

Keywords: networking protocol, satellite network, lossy channels, SPDY, HTTP, performance evaluation.

1 Introduction

Nowadays, services for sharing personal contents with a high degree of interactivity are considered new real killer applications of the Internet. This new paradigm has been mainly applied to the World Wide Web (WWW), which now approximates the *Social Web* initially envisaged by the World Wide Web Consortium (W3C). However, instead of a unified design, its implementation is based on heterogeneous sets of specific platforms, often with overlapped functionalities. We mention, among the others: blogs, wikis, Online Social Networks (OSNs), multimedia sharing platforms, and real-time collaboration frameworks.

In parallel, *legacy* websites have been enriched with such functionalities too. In fact, about the totality of *modern* webpages has some features to support interactivity or to provide dynamic contents. To this aim, one of the most popular approaches relies on the *mash-up* technique enabling to retrieve and merge data

R. Dhaou et al. (Eds.): PSATS 2013, LNICST 123, pp. 123–134, 2013.

from different remote providers (see, e.g., reference [1] for a paradigmatic example on the composition of mashable information). Moreover, the social vocation of the Web has been also ported to plain websites, e.g., via third party plug-ins directly embedded in the HTML. In order to be effective, interactivity needs a constant data exchange between the involved endpoints. A popular approach is to use the Asynchronous JavaScript and XML (AJAX) paradigm, which constantly transmits data between the server and the client over an indefinitely held HTTP connection. This also requires pages to embed proper scripts (e.g., the `XMLHTTPRequest` Javascript object), or additional software components to interact with remote services (often defined as plug-ins).

Such aspects lead to the so-called *Web 2.0* also accounting for mutations in the structure of pages, and characteristics of the related Internet traffic. In more details, a Web page is composed of many *objects*, which have to be retrieved to compose the whole content, i.e., the *main* object containing the HTML code, and (multiple) *in-line* objects(s) linked within the hypertext. The Web 2.0 heavily alters the characteristics of in-line objects, which can now embed additional services, such as for audio/video streaming, or to interact with an OSN [2]. Nevertheless, this enriched vision has not been limited only to contents, since it is now possible to deliver full-featured applications via a Web page. As an example, many Software-as-a-Service (SaaS) platforms [3] are based on interactive Graphical User Interfaces (GUIs) directly operated from the browser. Yet, to issue commands and provide feedbacks to users, they require a non-negligible amount of bandwidth and real-time constraints for assuring prompt data synchronization with a remote back-office. To summarize, the highly interactive nature of Web 2.0 reduces the accuracy of the *page-by-page* model.

Another important feature to comprehensively evaluate the modern Internet is its increased support of *mobility*. Thus, many network appliances also assure connectivity via wireless links, e.g., IEEE 802.11, the Universal Mobile Telecommunication System (UMTS), Long Term Evolution (LTE), and satellite channels. Yet, mobile nodes impose constraints clashing with the resource consuming nature of the Web 2.0. This is even truer in the case of satellite communications, potentially leading to additional hazards in terms of performances, for instance due to delays (see, e.g., reference [4] for a performance evaluation of an OSN accessed through a geostationary satellite facility). We point out that a satellite link is not only deployed to support mobile nodes, but it is also the main choice to grant access to the Internet in rural areas, or developing Countries.

Hence, the more aggressive behaviors of Web 2.0 applications also need proper adjustments in the protocol stack, especially for the case of the HTTP. In this perspective, a cutting-edge solution is SPDY [5], i.e., an enhanced HTTP supporting data compression and connections multiplexing. It can also mitigate the impact of channels with large Round Trip Times (RTTs) and high packet losses. According to reference [5], when used on wired links, SPDY can reduce download times in the range of 27-60%. With such premises, it could be a very suitable solution for improving performances when accessing the Web from a satellite network. Nevertheless, since SPDY uses a single transport connection, its deployment in

satellite networks can lead to further benefits. For instance, typical transport-layer enhancements like Performance Enhancing Proxies (PEPs) can experience a reduced workload, i.e., in terms of TCP connections to be handled to serve multiple Web sessions.

From the author's best knowledge, there are not any prior attempts to characterize SPDY over satellite channels. Thus, this paper evaluates the effectiveness of using SPDY in place of standard HTTP to increase performances of Web access over satellite networks. The contributions of this work are: *i*) to understand the most relevant behaviors of the SPDY protocol when used over heterogeneous wireless networks, especially those using large *delay* channels such as satellite one; *ii*) to showcase the creation of an ad-hoc testbed, which can be reused for similar investigations; *iii*) to provide an earlier comparison between HTTP and SPDY when used to access some popular websites.

The remainder of the paper is structured as follows: Section 2 compares HTTP and SPDY, while Section 3 showcases the testbed and some basic patterns of the protocol. Section 4 deals with the performance evaluation when in a worst case scenario, and Section 5 concludes the paper.

2 Evolving from HTTP to SPDY

As hinted, the growing complexity of Web 2.0 should be also properly addressed by the protocol architecture. Some issues have been partially resolved by amending the HTTP protocol specification, namely: *i*) to avoid performance degradations, the increased number of objects composing a page requires proper parallelization of the retrieval process; *ii*) the rising volume of data needs to increase the protocol efficiency, also by reducing overheads; *iii*) the "distributed" nature of many contents (i.e., data can be stored in different providers), jointly with the need of long-held connections for interactivity purposes, require more flexible policies. To partially fulfill requirements *i*) – *iii*), the HTTP/1.1 [6] relies on multiple connections for concurrency. Even so, this can introduce additional hazards, including: supplementary round trips for completing the connection setup/teardown phases, delays in the slow-start phase of the TCP, as well as connection rationing by the client, e.g., when it tries to avoid opening too many connections over a single server. HTTP/1.1 also uses pipelining to send multiple requests over a single TCP connection without waiting for a response. This technique limits the offered load in terms of TCP Protocol Data Units (PDUs), and can also reduce the loading times of pages. Potential benefits are greater over high latency connections, such as satellite Internet connections [7]. Figure 1 shows how pipelining reduces the connection time with the respect to sequential HTTP requests.

However, the gains of pipelining are limited by the HTTP/1.1 protocol specification, since the server must generate responses ordered as the requests were received. Thus, the entire flow of information belonging to a connection is ruled according to a first-in-first-out policy. In turns, this can lead to performance

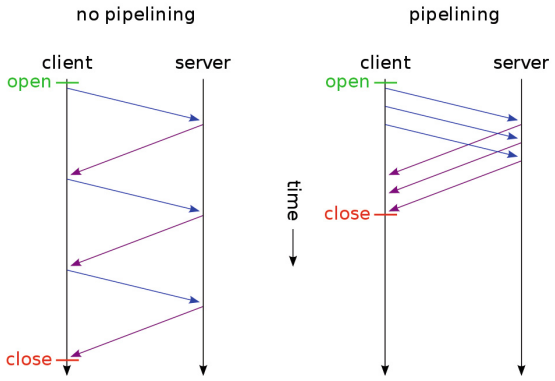


Fig. 1. Example of pipelined HTTP requests

degradation due to Head of Line (HOL) blocking[1] phenomena. Unfortunately, HTTP pipelining requires to be implemented both within the client and the server. As today, it is not widely available into existing browsers.

To prevent similar issues, SPDY introduces a specific *framing layer* (also named session layer) [5] for multiplexing concurrent streams atop a single persistent[2] TCP connection, as well as any other reliable transport service. Furthermore, it is optimized for HTTP-like request-response conversations, and also guarantees full backward compatibility with the plain HTTP.

In more details, SPDY offers *four* major additional improvements to the network behavior of HTTP:

1. *multiplexed requests*: to increase possible gains, the SPDY protocol specification does not impose any limits to the number of concurrent requests that can be sent over a single connection;
2. *prioritized requests*: to avoid congestion phenomena due to scarce resources at the network level, clients can indicate resources to be delivered first. This can enhance the Quality of Experience (QoE) of a service, even in presence of incomplete pages;
3. *compressed headers*: modern Web applications force the browser to send a significant amount of redundant data in the form of HTTP headers. Since each Web page may require up to 100 sub-requests, the benefit in term of data reduction could be relevant;

[1] HOL blocking is a performance-limiting event occurring when a trail of PDUs is held-up by the first packet. This can happen when in presence of network switches with buffered inputs, protocols supporting out-of-order delivery, or multiple requests as in the case of HTTP pipelining.

[2] An HTTP persistent connection, or HTTP keep-alive, uses a single TCP stream to move multiple HTTP requests/responses, instead of opening a new connection for each single request/response pair.

4. *server pushed streams*: this feature enables content to be pushed from servers to clients without additional requests.

Though, mechanisms 1) – 4) are somewhat analogous to HTTP pipelining, thus leading to potential transport level HOLs. This is even truer when in presence of packet losses, which could invalidate compression and prioritization as a consequence of TCP error recovery strategies. For such reasons, SPDY needs a proper comprehension when in jointly used with error prone links.

For what concerns all the protocol resources (e.g., documentation and software), they are provided by the SPDY Google Developer Group. In addition, performance evaluations in real-world use cases have been by the Chromium Projects[3], which spawned the "*Let's make the Web faster*" initiative[4]. The preliminary results were focused on comparing SPDY against HTTP. The reference testbed has been developed by simulating a user population browsing, from a Small Office Home Office (SOHO) Internet access, a selected set of reference Web sites called the "*top 100*". Additionally, different packet losses have been considered, i.e., in the range of $0 - 1\%$. The main outcome is that SPDY reduces the average page load times, for 25 websites of $27 - 60\%$ when using the TCP without the Secure Socket Layer (SSL), and $39 - 55\%$ when SSL is in place. A similar set of trials, also devoted to quantify the impact of SPDY over mobile terminals, has been proposed in reference [8], where the authors ran experiments on Chrome for Android[5], in order to have a draft 2 version of the protocol. Also in this case, SPDY outperformed HTTP by assuring an average reduction of the 23% in terms of loading times of pages.

Yet, literature still lacks of a thorough performance analysis of SPDY, both in terms of precisely quantifying the benefits for each feature introduced, and over a wide variety of network scenarios. Despite that, we decide to focus on the multiplexing ability of SPDY when used over a satellite channel. On one hand, this effort emphasizes benefits of exchanging data within a single SPDY stream tunneled within a TCP connection. On the other hand, the trials underline the impact of header compression when juxtaposed over a network having high access delays and packets losses. Additional benefits, compared with the current state-of-the-art literature, are the increased understanding of the HOL/error relationship, as well as the consequences of large propagation delays over the congestion control of the TCP.

3 Testbed Creation and Measurement Methodology

As said, our goal is to comprehend and evaluate the basic behaviors of SPDY when used in a satellite environment. In order to assure a fair approach, we test the protocol against a subset of the "top 100 websites" list compiled by Google. To this aim, we implemented a testbed composed by: an SPDY-enabled

[3] `http://www.chromium.org/chromium-projects`
[4] `http://www.chromium.org/spdy/spdy-whitepaper`
[5] `http://www.google.com/intl/en/chrome/android/`

browser (i.e., Google Chrome), which has been properly scripted for automating the content retrieval, as well as data collection; an emulated satellite link; a proxy for accessing non-SPDY sites; additional tools, such as a traffic sniffer.

To emulate the satellite access, we used `netem`[6], which is part of the native Linux queuing discipline. It permits to easily superimpose wide area network characteristics, e.g., the delay and the packet error rate, over standard routing strategies. Since `netem` can only process inbound packets, as a quick workaround, we created a new Intermediate Functional Block (IFB) pseudo device, in order to handle the emulation discipline also to incoming packets. Another possible solution would be using `netem` both in the satellite gateway and terminal. As regards the proxy, it has been deployed to cope with the scarcity of SPDY enabled websites on the Web. Hence, we used a `node.js` SPDY server[7], which has been configured to act as a Web proxy, as depicted in Figure 2.

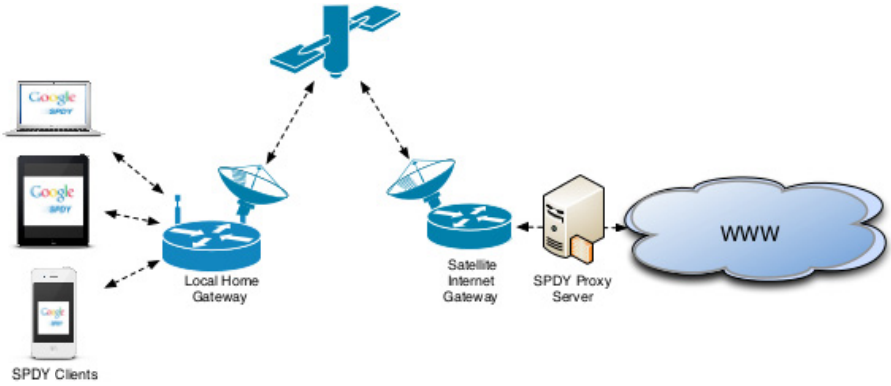


Fig. 2. Reference architecture of the adopted tested

3.1 Testbed Validation and Basic Protocol Understanding

As a consequence of a lack of thorough past investigations on the SPDY protocol, this initial round of tests has been performed to comprehend some of its core behaviors, also to understand wether SPDY could really outperform HTTP when used over a satellite link. Therefore, we focus on a GEO satellite system, that has been emulated by imposing a round trip time (RTT) of 720 ms (according to real measurements performed on the Skyplex Platform as discussed in reference [9]), and by limiting the bandwidths to 1 Mbit/s and 256 kbit/s, in the forward and return links, respectively. To have a controlled environment, in this first run of tests we assumed the channel as error-free.

The first analysis compares HTTP against SPDY in terms of used transport connections. This metric is a rough "complexity" indicator, which quantifies the perspective reduction of overheads for enhancing/splitting TCP flows, for instance by using a PEP machine. Table 1 summarizes the number of transport

[6] `http://swik.net/netem`
[7] `https://github.com/igrigorik/node-spdyproxy`

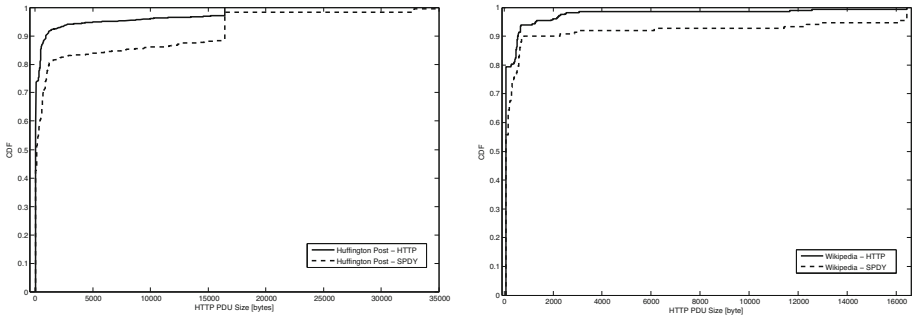
Table 1. Number of TCP connections per site when using standard HTTP

Site name	# of connections
Flickr	14
Huffington Post	173
Reddit	41
Microsoft	58
Slashdot	50
Wikipedia	17
BBC	88

connections required to download the contents of the selected sites when using standard HTTP. As reported, the presence of an extremely high amount of TCP conversations is mainly due to the content-richness nature of Web 2.0 applications, as well as the need of retrieving a composite set of objects, e.g., plug-ins or multi providers mash-ups. Yet, Table 1 underlines that not all sites present such extreme characteristics. This is the case of Wikipedia, which is mainly delivered through a text-based layout without any additional service embedded (e.g., advertisements, Facebook plug-ins or location widgets à-la Google Maps).

On the contrary, when pages are retrieved through the SPDY proxy, the amount of required TCP connections always reduces to *four*. This is a consequence of the multiplexing architecture of the protocol. Besides. we underline that only *one* connection is strictly related to SPDY traffic, while others are initiated by the browser to perform navigation statistics, or to fetch data for remote services (e.g., to provide users search suggestions/completions). Unfortunately we were not able to inhibit such process. Yet, we were able to precisely quantify the resulting overhead as to avoid "noise" in the collected results. Specifically, despite the adopted protocol, only one connection generates traffic and produces less than 100 kbyte of data. The other two connections simply perform a `SYN/FIN` exchange, probably to enable some kind of viewing time profiling. Thus, we solely focus on the transport connection devoted to transfer the Web page, which can be always correctly identified. As a consequence, the adoption of SPDY leads to a very minimal load in terms of connection to be managed, also accounting for a reduced complexity. Since satellite networks often use some kind of middleboxes (e.g., PEPs) for increasing the throughput of transport layer protocols, SPDY has to be considered a very interesting option to shift overheads at the borders of the network.

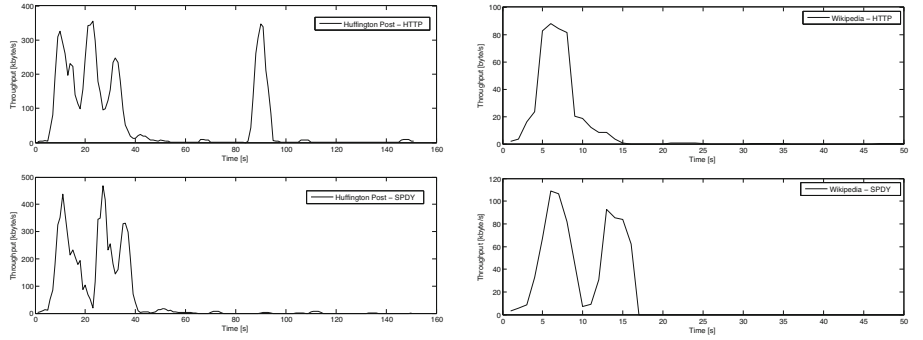
Another benefit is given by the compression of data via the `gzip` algorithm accounting for relevant gains when in presence of textual information, e.g., large headers. To better quantify this improvement, Figure 3 shows the Cumulative Density Function (CDF) of the HTTP PDUs produced when using HTTP and SPDY, when accessing two popular websites. Since both the client and the SPDY proxy operate on the same virtual machine in loopback, the maximum allotted PDU can benefit of a Maximum Transmission Unit (MTU) up to 64KB.



(a) CDFs of Huff-Post PDUs. (b) CDFs of Wikipedia PDUs.

Fig. 3. CDF of the HTTP and SPDY PDUs for two reference Websites

As depicted, SPDY optimizes the PDU size, with the acceptation that small packets are fewer than in the HTTP case. Consequently, the traffic is less fragmented, thus reducing the performance issues related to the TCP throughput. Similar results are observable when accessing a text-based service likes Wikipedia. For such a reason, the behavior of CDFs are different than when in presence of content-rich sites, but inspecting the throughput will reveal enhancements also in this case.



(a) Throughputs for the Huff-Post. (b) Throughputs for Wikipedia.

Fig. 4. Throughput when retrieving the Huffington Post and Wikipedia homepages with and without SPDY

Figure 4 shows improvements in terms of throughput, leading to a significant reduction of the downloading time. We point out that traffic spikes are periodically generated even when the page has been completely received for updating in a real-time manner some portion of the page (i.e., it represents exchanges due to AJAX-like techniques). Instead, for the case of Wikipedia the gain is less obvious, primarily due to effect of the aggressive compression performed by `gzip` over

the textual components composing the page. Yet, apart from a small amount of data transmitted in the $20 - 25$ s range, the largest part of the information is received almost faster when in presence of HTTP.

4 Performance Evaluation over Satellite Accesses

In this section, we evaluate the robustness of SPDY into account a worst-case scenario, i.e., we want to analyze some "macro" effects revealed by extreme conditions like a high Bit Error Rate (BER). We point out that we do not assume the presence of a TCP splitting architecture to isolate the satellite portion of the network. In fact, we are interested into understand wether SPDY can be used as a standalone workaround. According to Figure 1 the client connects to the satellite gateway through an IEEE 802.11 wireless link. To this aim, we used `netem` to emulate a bi-stable wireless channel, as depicted in Figure 5. The model is implemented via a 2-state Markov process, which is characterized by the transition probabilities p and q. Specifically, p is the probability of passing from the error-free state `0`, to the error-prone state `1`, while q quantifies the vice-versa. According to a past measurement campaign (see reference [10] for further details), we set $p = 0.54$ and $q = 0.78$, as to characterize a wireless indoor channel with an average packet loss (defined as *ploss* in the following) of 6.8%.

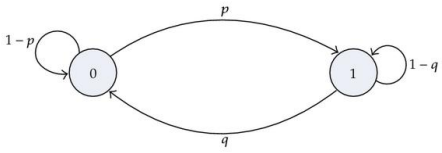
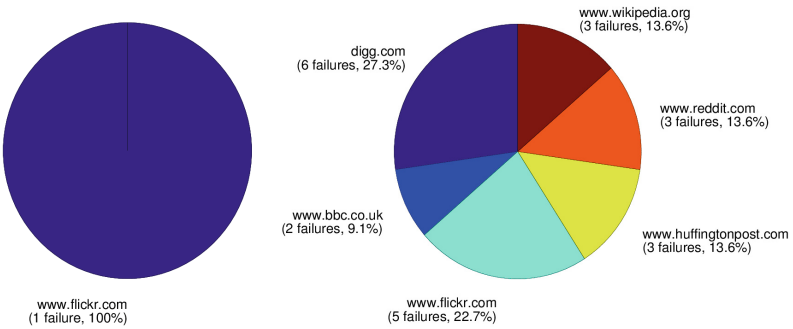


Fig. 5. Model of an on-off bi-stable channel

To conduct the evaluation, we used eight reference sites, which have been repeatedly accessed more than 20 times, as to have steady results. The same usage-pattern has been applied both for the cases of HTTP and SPDY. Figure 6 shows the distributions of load page failures for each protocol. For what concerns relative percentages, we experienced that SPDY fails 22 times, i.e. the 12.5%, while the HTTP only fails once, i.e., the 0.57%. Hence, SPDY appears to be more fragile when in presence of errors. To better elaborate on this point, we discovered that page failures mainly happens when the browser cannot correctly receive the `index.html` (with the acceptation of the main HTML body). This is a consequence of SPDY transmitting the whole content via a single TCP connection, rather than using multiple streams as the HTTP. Therefore, a single connection failure (even in the set-up phase) could block the page request transaction in its entirety.

Collected traffic traces reveal some duplicated `SYN-ACKs` received by the client side, eventually causing the conversation fail. In this perspective, Figure 7 reports a paradigmatic example. Another point to be remarked is that a `SYN` packet is



(a) HTTP (b) SPDY

Fig. 6. Failures when retrieving a page with a $ploss = 5\%$

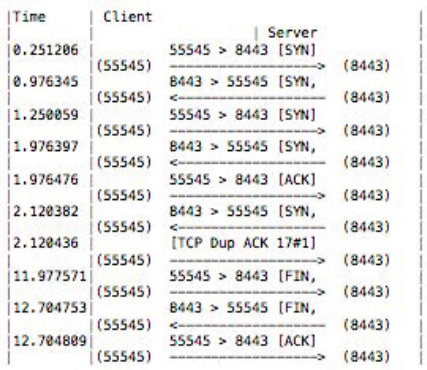


Fig. 7. Example of a failed TCP conversation

sent twice after the expiration of the Retransmission Time Out (RTO) which is set to 1s. This is an outcome of having the average RTT set to 720 ms, as not the unique delay imposed by the network. In fact, it does not include timings due to internal data processing/percolation for each element composing the network, as well as TCP buffering traversal. Thus, the overall delay could be greater than the RTO threshold. However, even though HTTP seems to be more robust against errors than SPDY, such behavior is almost uninfluential. In fact, failures cannot be able to capture implications on the perceived user experience, thus they must be compared against loading times. In this vein, Table 2 and 3 showcase collected timing statistics, both for HTTP and SPDY. According to data, on the average, SPDY seems to slightly outperform HTTP. Even if HTTP fails in less occasions than SPDY, the times to retrieve a content are excessive. As a consequence, a user would close the browser (i.e., abandon the session) before the complete reception of the page. Therefore, paying a price in terms of failed receptions would lead to a better QoE, rather than waiting for too long periods for accessing a content.

Table 2. HTTP loading time

Site name	min	max	avg
Flickr	26.15	372.48	160.94
Huffington Post	120.64	637.29	351.11
Reddit	49.84	584.11	261.59
Microsoft	143.63	552.00	298.38
Slashdot	107.52	544.95	286.66
Wikipedia	38.28	344.54	135.36
BBC.co.uk	137.07	769.17	382.40

Table 3. SPDY loading time

Site name	min	max	avg
Flickr	52.89	258.08	141.89
Huffington Post	263.54	669.49	362.00
Reddit	41.69	332.26	124.83
Microsoft	144.43	593.04	296.25
Slashdot	148.43	364.40	211.00
Wikipedia	13.38	184.10	48.98
BBC.co.uk	175.93	468.78	270.60

To clarify the QoE perception, we would recall the worst-case nature of the test. SPDY could lead to a barely satisfactory experience when accessing the Web, even in presence of an unacceptable average *ploss*. SPDY is more robust against high BERs and allows slightly reduced loading times, as it can be observed by comparing Table 2 and 3. Nevertheless, quantifying only loading times could be misleading, since with SPDY a page could be almost readable after ∼30 s, even if its completion could happen in many minutes.

5 Conclusions and Future Work

In this paper we investigated the use of SPDY to enhance performances when retrieving Web contents over an heterogeneous wireless scenario composed by an error-prone IEEE 802.11 access and a satellite link. Then, we showcased the creation of an ad-hoc testbed, and we also provided a basic understanding of the SPDY protocol compared to HTTP when jointly used with a satellite link. We investigated the effect of the packet loss on the overall performance, especially in terms of page loading time, and loading failures. As a result, SPDY is a promising protocols, since it outperformed HTTP in our tests, while reducing the complexity in terms of number of transport connections.

Future work aims at enriching the experimental results, also by testing SPDY with a more complete variety of channel conditions. Besides, part of our ongoing research deals with the creation of a more precise emulated environment. In

particular, to test SPDY when the satellite links is implemented through a DAMA systems as the one discussed in reference [9].

Acknowledgment. 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 N. 23089/10/NL/CLP.

References

1. Zhang, J., Karim, M., Akula, K., Ariga, R.K.R.: Design and development of a university-oriented personalizable web 2.0 mashup portal. In: Proceedings of the 2008 IEEE International Conference on Web Services, ICWS 2008, pp. 417–424. IEEE Computer Society, Washington, DC (2008)
2. Caviglione, L.: Extending http models to web 2.0 applications: The case of social networks. In: Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing, UCC 2011, pp. 361–365. IEEE Computer Society, Washington, DC (2011)
3. Knorr, E.: Software as a Service: The Next Big Thing (2009), http://www.infoworld.com/article/06/03/20/7610312FEsaas1.html
4. Caviglione, L.: Can satellites face trends? The case of web 2.0. In: Int. Workshop on Satellite and Space Communications, IWSSC 2009, pp. 446–450 (2009)
5. Belshe, M., Peon, R.: SPDY protocol - draft 3 (2012), http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3
6. 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
7. Nielsen, H.F., Gettys, J., Baird-Smith, A., Prud'hommeaux, E., Lie, H.W., Lilley, C.: Network performance effects of HTTP/1.1, CSS1, and PNG. SIGCOMM Comput. Commun. Rev. 27, 155–166 (1997)
8. Welsh, M., Greenstein, B., Piatek, M.: SPDY performance on mobile networks (2012), https://developers.google.com/speed/articles/spdy-for-mobile
9. Gotta, A., Potorti, F., Secchi, R.: An analysis of tcp startup over an experimental dvb-rcs platform. In: 2006 International Workshop on Satellite and Space Communications, pp. 176–180 (2006)
10. Barsocchi, P., Bertossi, A.A., Pinotti, M.C., Potorti, F.: Allocating data for broadcasting over wireless channels subject to transmission errors. Wireless Networks 16, 355–365 (2010)