EAI Endorsed Transactions
on Mobile Communications and Applications
Research Article ICST.ORG

Characterizing SPDY over High Latency Satellite Channels

Luca Caviglione[1,*], Alberto Gotta[2]

[1]Institute of Intelligent Systems for Automation (ISSIA), National Research Council of Italy (CNR)
[2]Information Science and Technologies Institute (ISTI), National Research Council of Italy (CNR)

Abstract

The increasing complexity of Web contents and the growing diffusion of mobile terminals, which use wireless and satellite links to get access to the Internet, impose the adoption of more specialized protocols. In particular, we focus on SPDY, a novel protocol introduced by Google to optimize the retrieval of complex webpages, to manage large Round Trip Times and high packet losses channels. In this perspective, the paper characterizes SPDY over high latency satellite links, especially with the goal of understanding whether it could be an efficient solution to cope with performance degradations typically affecting Web 2.0 services. To this aim, we implemented an experimental set-up, composed of an ad-hoc proxy, a wireless link emulator, and an instrumented Web browser. The results clearly indicate that SPDY can enhance the performances in terms of loading times, and reduce the traffic fragmentation. Moreover, owing to its connection multiplexing architecture, SPDY can also mitigate the transport layer complexity, which is critical when in presence of Performance Enhancing Proxies usually deployed to isolate satellite trunks.

Received on 27 January 2014; accepted on 27 July 2014; published on 28 December 2014

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

doi:10.4108/mca.2.5.e3

1. Introduction

The Internet is daily used to share personal contents, and to exchange information in an highly interactive fashion. Eventually, such paradigms greatly mutated the Web. To offer a proper degree of dynamism, also legacy websites have been enriched with sophisticated features, e.g., the Asynchronous JavaScript and XML (AJAX) to exchange data between the server and the client over an indefinitely held HyperText Transfer Protocol (HTTP) connection. Besides, pages no longer have clear boundaries, since they embed contents usually *mashed-up* from several remote sources [1]. Such features are usually grouped under the *Web 2.0* umbrella term, and heavily altering the characteristics of *in-line objects*, i.e., those linked against the *main* object containing the hypertext [2]. This enriched vision is not only limited to pages, since full-featured applications can be accessed from the browser, as it happens in Software-as-a-Service (SaaS), where a non-negligible amount of bandwidth and real-time guarantee are needed to enable data synchronization and proper feedbacks. As a consequence, the highly interactive nature of Web 2.0 reduces the accuracy of the *page-by-page* model, as well as the performances of protocols built on such template.

In parallel, the Internet is not only more *interactive*, but also more *mobile*, leading to a dramatic increase in the usage of wireless links, e.g., the IEEE 802.11, the Long Term Evolution (LTE), and satellite channels. The latter are still the main choice to access to Internet in rural areas or developing countries. Yet, mobility, high error rates and delays impose constraints clashing with the resource consuming nature of the Web 2.0. This is even truer in the case of satellite communications, often leading to additional hazards in terms of performances [3].

In this scenario, the Web 2.0 demands adjustments in the HTTP. Specifically: *i*) the increased amount of in-line objects requires a parallelization of the retrieval process, and *ii*) the rising volume of data needs reducing the protocol overheads. As a partial workaround, HTTP/1.1 [4] relies on multiple connections, even if this can lead to additional hazards, such as increased Round Trip Times (RTTs) to setup/teardown a connection, bigger delays in the TCP slow-start phase, and connection rationing phenomena (i.e., the browser cannot open too many connections over a single server). Such unwanted effects are further emphasized by delays characterizing satellites, then imposing to investigate new mechanisms.

*Corresponding author. E-mail: luca.caviglione@ge.issia.cnr.it

To this aim, SPDY (pronounced "SPeeDY") could be a very suitable solution to accessing Web 2.0 from a satellite network, since it is an enhanced HTTP supporting header/data compression and connections multiplexing [5]. For instance, when used on wired links, it can reduce download times in the range of 27-60% [5]. Also, by funneling all the data into a single TCP flow, it can mitigate the impact of large RTTs, high packet losses, and it further simplifies the produced traffic. Nevertheless, Performance Enhancing Proxies (PEPs) can experience a reduced workload, i.e., in terms of states to be handled when serving a vast user population.

However, when used over 3G cellular networks, SPDY does not show any clear performance improvements, specially due to the complex cross-layer interactions [6]. Close conclusions are showcased in [7], even if the Authors underline that in simpler architectures, the header compression introduced by SPDY enables benefits. A comparable set of trials has been proposed in reference [12], where the authors ran experiments on Chrome for Android, in order to have a *draft 2* version of the protocol. In this case, SPDY outperformed HTTP by assuring an average reduction of 23% in terms of loading times of pages. As regards possible architectural enhancements, in reference [8] a protocol and caching infrastructure to improve performance in multi-domain and mobile scenarios is proposed. To the author's best knowledge, the only prior work characterizing SPDY over satellite is available in references [9] and [10], both clearly indicating that SPDY brings relevant benefits.

Therefore, this paper widely extends the work [9], and its main contributions are: *i*) to characterize the main behaviors of SPDY when used over heterogeneous satellite networks; *ii*) to showcase the creation of an ad-hoc/reusable testbed; *iii*) to provide an earlier comparison between HTTP and SPDY when used to access popular sites.

The remainder of the paper is structured as follows: Section 2 introduces SPDY. Section 3 discusses the testbed and the measurement methodology, and Section 4 showcases the results of the investigation. Lastly, Section 5 concludes the paper and proposes future research directions.

2. From HTTP 1.1 to SPDY

As said, to cope with the additional complexities of Web 2.0 applications, the HTTP protocol specification has been partially amended over the years. In details, in its last incarnation – the HTTP/1.1 [4], it relies on multiple connections to increase the concurrency of the process of retrieving objects. Also, HTPP/1.1 uses pipelining to send multiple requests over a single TCP connection without waiting for a response. This limits the offered load in terms of TCP Protocol Data Units (PDUs), and reduces the Page Loading Time (PLT) over satellite Internet connections [11]. However, achievable gains are limited by the protocol specification, since the server must generate responses in the same order of requests. Thus, the flow of each connection is ruled according to a first-in-first-out policy. In turns, this can lead to Head of Line (HOL) blocking, where the first packet locks an entire line. Besides, HTTP pipelining is still optional, and requires to be implemented within both the client and the server. As today, it is not widely available into existing browsers.

To prevent similar issues, SPDY introduces an ad-hoc *framing layer* (also named *session layer*) [5] to multiplex concurrent streams atop a single persistent TCP connection, as well as any other reliable transport services. Besides, SPDY offers a `settings` session-wide message enabling a proper negotiation of transport parameters between endpoints, e.g., to report the size of the Initial Congestion Window (ICW) of the TCP to the remote server. Furthermore, it is optimized for HTTP-like request-response conversations, and also guarantees full backward compatibility with HTTP. In more details, SPDY offers *four* major additional improvements compared to HTTP:

1. *multiplexed requests*: to increase possible gains, SPDY 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 in-line objects 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;

4. *server pushed streams*: this feature enables objects/resources to be pushed in advance from servers to clients without additional requests.

However, mechanisms 1) – 4) could be partially voided by HOL blockings at the transport level. This is even truer when in presence of packet losses triggering the error recovery strategies of the TCP, which could invalidate compression and prioritization. For such reasons, SPDY needs a proper comprehension when 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 partially performed within the framework of the Chromium Projects[1], which spawned the *"Let's make the Web faster"* initiative[2].

3. Description of the Testbed

In order to evaluate the behavior of SPDY over satellite links, we performed a set of trials using an emulated satellite environment, as depicted in Figure 1. To capture all the needed descriptive features and perform repeated trials, we developed an ad-hoc *client*, running on an Intel Core 2 T7400 at 2.16 GHz with Linux Ubuntu 12.10 64bit OS (kernel 3.2.0.37). To generate requests, we used an instrumented Google Chrome browser, which automatically loads URLs, and starts proper supporting scripts, for instance to capture data via `tcpdump`. We point out that we used Google Chrome since it natively supports SPDY, and offers a very comprehensive set of debugging features that allow placing proper hooks to log data in order to reconstruct timing statistics. To store such temporal traces, we used the HTTP Archive (HAR) file format, i.e., a JavaScript Object Notation (JSON) document collecting a variety of statistics, such as the page size, and timing of objects as well. Repeated trials have been automatized via proper shell scripts, and a `Node.js` module.

Another important component of our testbed is the SPDY *proxy*. Also in this case we used the same Linux version of the client, and it has been developed with `Node.js`. The proxy has not been only deployed to access non-native SPDY sites, but also to increase the accuracy of our measurements. In fact, many Web 2.0 websites provide contents that dynamically change over time (e.g., due to advertisements or comments performed by other users), even in between two adjacent requests, thus leading to measurements that cannot be compared. Therefore, the contents used to investigate SPDY have been cached by using the `Web Page Replay` tool. In this manner, we relied on a set of websites with deterministic behaviors. Moreover, this approach can also avoid possible additional noises introduced by the Internet, such as congestion phenomena at the server side, or in some portions in the public network.

To emulate the *satellite link*, we used `netem`, which is now 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.

The TCP Initial Window (`IW`) is set to 10 segments by default, while we disable the `TCP Slow-Start Restart After Idle`, as suggested by Google and many past research works [13] as to improve the responsiveness of the proxy.

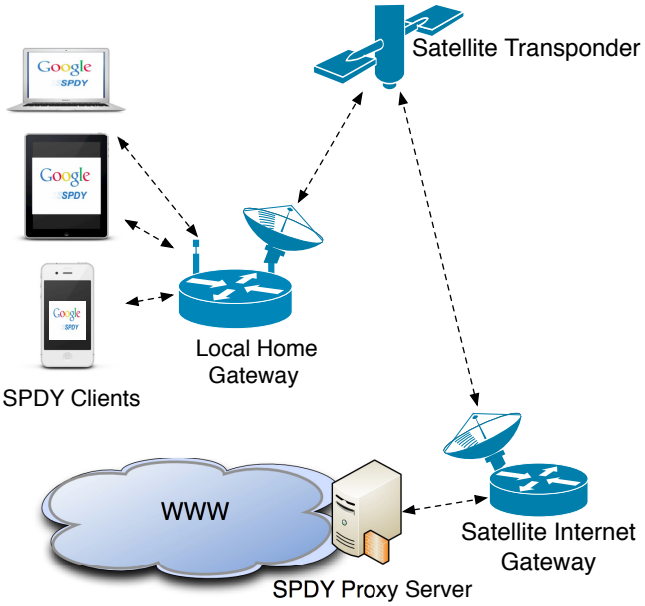

Figure 1. Reference architecture of the adopted tested. The satellite link has been emulated with different values of RTT and packet loss.

To have realistic data, we used a subset of the "top 100 websites" list compiled by Google. To have a small but composite set of characteristics, we shortlisted the following sites:

- Reddit and Slashdot: they rely on a limited amount of large images, but use a variety of small graphical elements;
- Wikipedia: it uses a simple/text-based layout, also reducing to the minimum the number of external dependencies;
- Huffington Post and BBC: they also embed plugins for video playback and to comment news. Both exploit a great number of inline objects scattered across different domains;
- Flickr and Microsoft: they include very large graphic elements, and plug-ins to implement carousels, i.e., they are archetypes of pages crafted for showcasing contents or products.

3.1. Preliminary Tests

As a consequence of the lack of thorough past investigations on SPDY, we performed an initial round

[1] `http://www.chromium.org/chromium-projects`
[2] `http://www.chromium.org/spdy/spdy-whitepaper`

of tests to better comprehend its core behaviors, as well as to validate our testbed. Therefore, we emulated a geostationary (GEO) satellite access, emulated by imposing the RTT of 520 ms, and by limiting the bandwidths to 1 Mbit/s and 256 kbit/s, in the forward link and return link, respectively (see, e.g., [14] for a discussion on tuning emulated satellite testbeds starting from real measurements). To have a controlled environment, in this first run of tests we assumed the channel as error-free. Each trial has been repeated 20 times.

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 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).

Table 1. Number of TCP connections per site when using standard HTTP. Ranking is performed according to Kb per page *on the wire* and to label sites in the rest of the paper.

Rank	Site name	# of conn.	Kb/Page
1	Wikipedia	17	111.13
2	Reddit	41	371.42
3	Flickr	14	499.28
4	Slashdot	50	712.62
5	BBC	88	950.73
6	Microsoft	58	1176.25
7	Huffington Post	173	1481.80

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 Layer4 connection transfering the Web page, which can be always correctly identified.

Hence, the adoption of SPDY results into a very minimal load in terms of the connections 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.

To complete the characterization of used sites, Table 2 reports the main features of each selected website, in order to understand some behaviors of HTTP in terms of page compositions. We also used the *Page Size* intended as Kb per site *on the wire* as a metric to rank sites (see, Table 1) to better organize the presentation of results in Section 4.

4. Characterization of SPDY

In this section, we compare SPDY and HTTP, by extending the emulated satellite environment presented. Specifically, we tested the protocols when using a Wireless LAN (WLAN) access, as to emphasize their robustness when in presence of errors. Therefore, as depicted in Figure 1, the client connects to the satellite gateway through a wireless link. Thus, we characterized the wireless impairment within the range $0 \div 1\%$ of the average packet loss (defined as *ploss* in the following). Also, we used two different values for the RTT, ranging from 520ms to 720ms, as to account for multiple use platforms. The bandwidth of the satellite link, the used sites and the usage patterns are the same previously discussed in Section 3.1.

Since both protocols have the goal of delivering contents to end users, investigating only the features of the produced traffic does not allow to precisely infer the user experience. In this perspective, we will also present results to better comprehend SPDY at the application level.

Lastly, for many metrics we will provide average values without a per-site granularity. This, will enable to offer a general idea on how SPDY could impact "globally" when used to browse the Web 2.0. Yet, to capture some more fine-grained details, we will compare Wikipedia against Huffington Post, since the former is content-rich, while the latter is essentially based on text and without any further multimedia or third-part plugin.

Table 2. Basic Statistics of Websites used as benchmark (average values).

Site	Reqs	Page Size [Kb]	N. Domains	Data/Reqs [Kb]	Reqs/Host [Kb]	Data/Host [Kb]
Wikipedia	18.97	131.36	3.00	6.94	6.32	43.79
Reddit	51.44	757.48	14.72	14.73	3.49	51.45
Flickr	17.37	968.46	4.24	55.79	4.10	228.56
Slashdot	48.76	1590.38	10.84	32.60	4.50	146.73
BBC	85.00	1523.36	12.00	17.92	7.08	126.95
Microsoft	52.43	1356.64	9.56	25.90	5.48	141.89
Huffington Post	110.11	3367.68	32.89	30.58	3.35	102.40

4.1. Analysis of the Packet Size

As a first step, we want to quantify the improvement of SPDY in terms of the usage of network resources. Thus, Table 3 and Table 4 showcase the analysis of the packet sizes for the two reference sites. In all the trials, SPDY offers a reduced number of tiny PDUs, if compared against the HTTP. This is one of the benefits given by the compression of data via the `gzip` algorithm. Coherently, the major gains are in the case of Wikipedia, where the huge amount of text, as well as a greater `content/header` ratio account for increased performances of the `gzip`.

Therefore, SPDY optimises the PDU size, by compressing HTTP header as well, with the acceptation that small packets are less frequent than for the HTTP case. Consequently, the traffic is less fragmented, also reducing the performance degradation of the TCP when transporting data over high *delay* × *bandwidth* channels. Such a behavior is even important in the case of satellite, since fewer packets reflect into less time spent in accessing the channel. In other words, high-delay links, even if with ad-hoc MAC protocols, will not experience loss of performances due to additional rounds of contention.

4.2. Analysis of Throughput

Another important aspect to characterize the usage of network resources of SPDY is given by the analysis of throughput. Yet, such a behavior does not efficiently capture how users perceive the "speed" at which a page is delivered. Still, it gives an idea on how the different protocols react against the high latency of satellite channels.

Table 5 presents a *vis-á-vis* comparison of HTTP and SPDY, when used to retrieve Wikipedia and Huffington Post in the different scenarios. In all the cases, both protocols experience higher throughputs when retrieving the Huffington Post, which is more content rich than Wikipedia. As a consequence, the underlying TCP has a longer temporal horizon to increase the transmission window and exploits the available resources (in some works such a behavior is also defined as the ability of the TCP in *"filling the bandwidth pipe*"). For the case of HTTP, latencies are partially compensated via its parallel connection flavor, while for SPDY, its steady state flow results in slightly reduced performances. However, as it will be explained in Section 4.4, the reduced number of TCP connections offers relevant gains in terms of loading times. When in presence of errors and high latencies, SPDY performs worse than HTTP mainly due to its more fragile nature rooted within the exploitation of a single connection. Such an aspect will be thoroughly evaluated in Section 4.5.

4.3. Page Loading Time

The Page Loading Time (PLT) is a widely used performance index, since it enables to partially understand how users perceive the performances of a given protocol.

Figure 2 showcases the waterfall graph for the Wikipedia site. The diagram has been produced by processing the `.har` archives collected during our tests. Specifically, it depicts precise timing statistics for each in-line object composing the page. Accordingly, the PLT is the time frame between the request of a page, and when the very last object is received by the browser (i.e., the page is complete).

Thus, the PLT is only a rough indicator, since it condenses a variety of temporal behaviors into a unique indicator, and it could fail to precisely represent the degree of the perceived Quality of Experience (QoE) perceived. Therefore, to effectively characterize SPDY, the PLT will be further detailed in Section 4.4, which presents *how* objects are delivered.

Figure 3 compares the different values of PLTs (the 95th percentile of the repeated trials averaged over the entire collection of considered sites) for both the HTTP and SPDY case. Let us discuss the case without losses. For the case of RTT=520 ms, SPDY grants smaller average times compared to HTTP. But, when the RTT=720 ms, such an enhancement vanishes, with HTTP performing better. Despite this, in both cases,

Table 3. Packet Size Analysis for Huffington Post – HTTP vs SPDY. Different trials are referred to (RTT [ms], ploss [%]).

HTTP				
Number of PDU	**(520, 0)**	**(520, 1)**	**(720, 0)**	**(720, 1)**
Total	2843.0	2864.8	2820.8	2896.2
< 80 bytes	1614.6	1605.6	1603.2	1629.2
80 - 1279 bytes	341.6	359.0	330.0	361.4
1280 - 1500 bytes	885.6	900.2	887.6	906.2
SPDY				
Number of PDU	**(520, 0)**	**(520, 1)**	**(720, 0)**	**(720, 1)**
Total	1797.2	1848.5	1804.6	1894.4
< 80 bytes	605.4	722.0	634.0	738.6
80 - 1279 bytes	197.2	149.5	172.4	167.6
1280 - 1500 bytes	995.2	977.0	998.2	988.2

Table 4. Packet Size Analysis for Wikipedia – HTTP vs SPDY. Different trials are referred to (RTT [ms], ploss [%]).

HTTP				
Number of PDU	**(520, 0)**	**(520, 1)**	**(720, 0)**	**(720, 1)**
Total	282.0	289.0	287.8	299.0
< 80 bytes	170.0	175.4	176.8	182.2
80 - 1279 bytes	56.0	57.2	55.6	58.4
1280 - 1500 bytes	56.0	56.4	55.4	57.4
SPDY				
Number of PDU	**(520, 0)**	**(520, 1)**	**(720, 0)**	**(720, 1)**
Total	175.0	179.4	177.4	181.8
< 80 bytes	77.6	80.8	78.6	83.6
80 - 1279 bytes	27.4	28.0	34.8	27.2
1280 - 1500 bytes	70.0	70.6	69. 8	71.0

Table 5. Throughput Analysis (in [kbit/s]) for Wikipedia and Huffington Post. Different trials are referred to (RTT [ms], ploss [%]).

	(520, 0)		**(520, 1)**		**(720, 0)**		**(720, 1)**	
Protocol	**HTTP**	**SPDY**	**HTTP**	**SPDY**	**HTTP**	**SPDY**	**HTTP**	**SPDY**
Wikipedia	149.40	135.40	89.40	123.80	117.00	111.80	97.00	89.40
Huffington Post	217.75	210.40	193.00	193.25	221.20	213.40	204.00	172.00

SPDY offers a better access to Web 2.0 contents , since the maximum values of PLTs are always smaller, also with a minimal range of variability.

For what concerns the case of ploss=1%, HTTP always exhibits reduced average times, compared to SPDY. To elaborate better on this point, when RTT=520 ms SPDY assures reduced variabilities, and smaller maximum time. Instead, when RTT=720 ms, timing ranges are similar. This behavior is due to the nature of HTTP of opening multiple parallel connections, thus enabling to better recover to burst of errors, as also showcased in terms of throughput. To summarize, besides from the viewpoint of PLT, SPDY is mainly impaired by its single-flow nature.

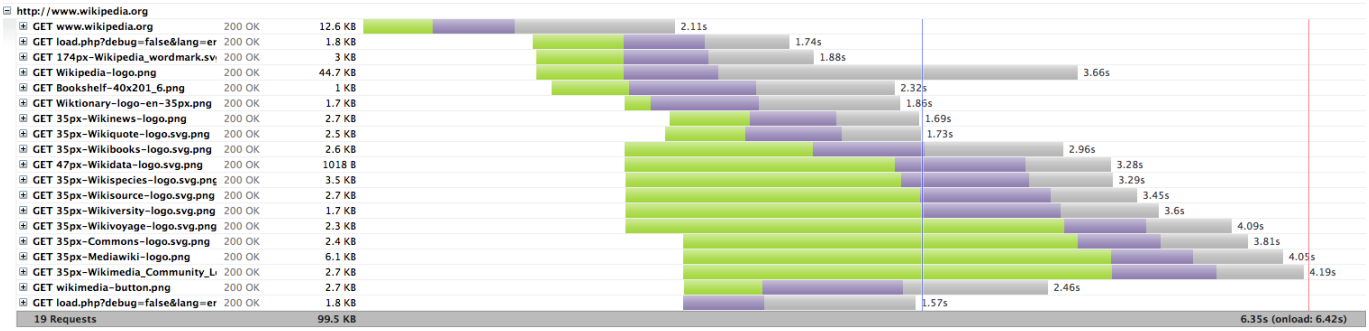

Figure 2. Waterfall diagram for the Wikipedia site.
(Color legend for times: *green* – *connect*, *purple* – *wait*, and *gray* – *receive*).

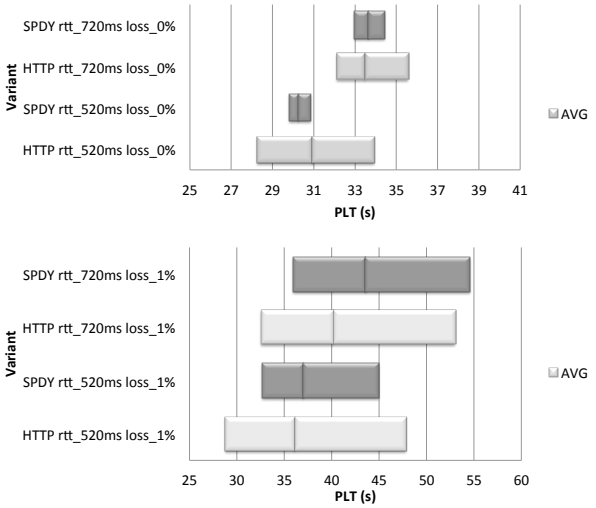

Figure 3. Average, maximum and minimum PLT values for all the sites used in the different trials. Values are in seconds.

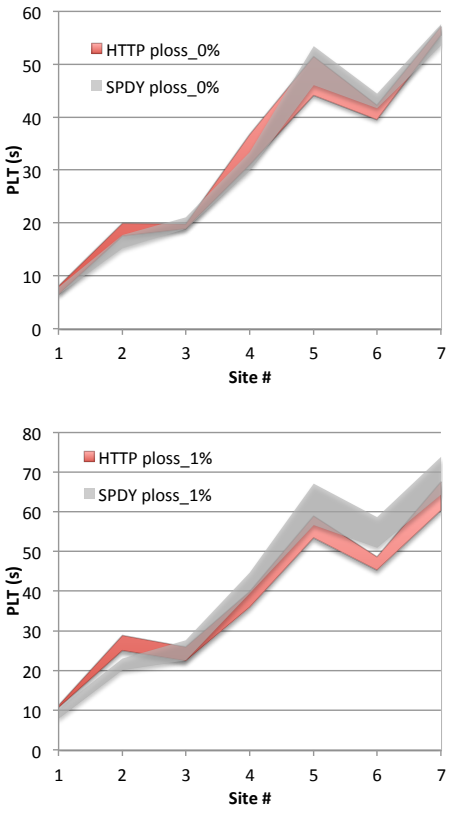

Figure 4. Per-site average PLT values. Sites have been numbered according the rank of Table 1 (Wikipedia is 1, and Huffington Post is 7).

Figure 4 offers a breakdown of performances in terms of PLT with a per-site granularity, which have been numbered according to the ranking of Table 1. The trends show the area ranging from minimum PLT values (when RTT=520 ms), and maximum one (when RTT=720 ms), for both the values of ploss.

For the error free case, the relation "the more bytes to transfer the more time to take" is not always true, e.g., for the Microsoft case (#6) the page design produces a short PLT, despite of a significant page size. Besides, in trials with losses, the errors lead to HOL blocking phenomena at the transport layer. In details, SPDY has a mixed behavior, since it outperforms HTTP on smaller sites, but its performance decreases on bigger ones.

4.4. Object Loading Time

As said, the PLT could fail to capture in a comprehensive manner the perceived user experience, since it does not give idea on how objects are received (hence, rendered) by the browser. Therefore, we characterize the performance of HTTP and SPDY with a per-object granularity, defining this metric Object Loading Time (OLT). To this aim, we introduce *three* timing parameters:

- *connect*: is the time required to establish a new connection with the server, also considering additional periods if the maximum number of

concurrent connections has been reached (six for the case of Google Chrome);

- *wait*: measures the timeframe between the delivery of the first HTTP request and the reception of the related HTTP header;
- *receive*: quantifies the time for retrieving all the objects composing the page.

As an example, Figure 2 can be also used to showcase a detailed breakdown of such times: *green* is the *connect*, *purple* is the *wait*, and *gray* is the *receive*.

Figure 5 compares HTTP and SPDY according to such metrics. Especially, when using SPDY, its single-stream architecture prevents additional connect times. Due to the delay sensitive three-way handshake of the TCP, SPDY dramatically reduces the time spent by the browser waiting for establishing a new connection, due to the re-use of the same TCP connection toward the same domain. It must be noted that by using a proxy (HTTP or SPDY) - a common practice for satellite scenarios - all the possible domains of a web page are masked by the proxy itself. The same applies for the teardown phase, where delays require more time to recover to *rationing* phenomena, i.e., the browser hits the maximum of parallel connections and has to wait the release of a socket.

As regards the receive time, it depends on how the available bandwidth is used. While the high delay heavily impacts over short-lived streams, the consistent use of a "steady state" TCP connection makes SPDY outperform again (i.e., via the `settings` features).

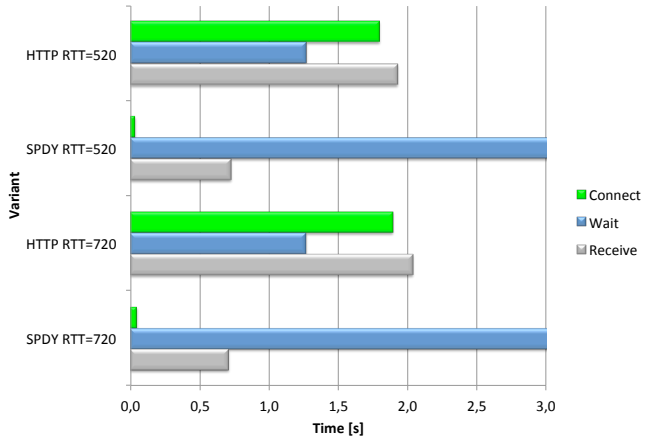

Figure 5. Breakdown of the average OLT on the overall dataset.

Instead, the wait parameter appears to be very critical for SPDY when used over satellite links. In fact, it results from the need of multiplexing many traffic flows into a single SPDY stream, as summarized in Figure 6. This introduces a bottleneck, mainly due to the lack of a priority-based scheduling policy within the implementation of the SPDY proxy.

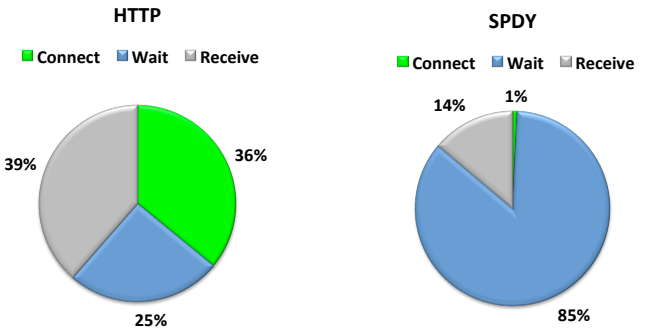

Figure 6. Percentages of the OLTs computed over the entire dataset.

Figure 6 deals with how OLTs are partitioned when the ploss = 0%. In details, it clearly indicates that when using HTTP the three parameters are almost equally distributed, while for the case of SPDY, the only further margin of improvement lays within reducing waiting periods. Therefore, it is highly advised to implement proper scheduling mechanisms as to avoid queuing/blocking issues due to serialization. Such an activity is part of our ongoing research. For the case of connect, similar outcomes have been collected when the ploss = 1%, while the wait and receive tend to increase due to error.

4.5. Worst Case Scenario Test

As presented, the single connection architecture of SPDY is its main source of fragility. Thus, we now evaluate its robustness by considering a worst-case scenario, i.e., a high Bit Error Rate (BER) deployment. This enables to better conclude whether SPDY can be used as a standalone workaround when acting without middleboxes mitigating the effect of an error-prone hop(s). For this round of tests, we the wireless accesses with an average packet loss of 6.8%. All the other parameters remain unchanged from the past round of tests, excepts websites, as it will be discussed.

Table 6 shows the distributions of load page failures for each protocol. For what concerns relative percentages, we experienced that SPDY fails 22 times, (12.5%), while the HTTP only fails once when retrieving Flickr (0.57%), confirming that SPDY is weaker than HTTP in error prone links. To better elaborate on this point, we discovered that page failures mainly happen when the browser cannot correctly receive the `index.html` (in the sense of the main HTML `body`). Therefore, a single connection failure (even in the set-up phase) could block the page request transaction in its entirety.

Table 6. Failures when retrieving a page with SPDY and a *packet loss* = 5%.

Site name	# of Failures	%
Huffington Post	3	13.6
Digg	6	22.7
BBC	2	9.1
Flickr	5	22.7
Reddit	3	13.6
Wikipedia	3	13.6

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. We point out that a SYN packet is 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.

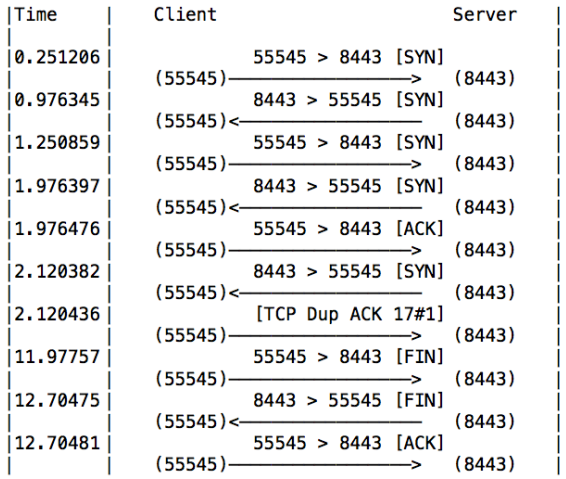

Figure 7. Example of a failed TCP conversation.

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 IEEE 802.11 access and a satellite link, characterized with different delays and BERs. We also showcased the creation of an ad-hoc testbed, as well as a basic understanding of the SPDY protocol compared to HTTP when jointly used with a satellite link. Hence, 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 protocol, since it outperforms HTTP in our tests, while reducing the complexity in terms of the number of transport connections.

However, the results clearly indicate that two main actions are needed to successfully exploit SPDY over satellites: *i*) a proper scheduling discipline is needed to reduce delays in the object delivery phase, and *ii*) some countermeasures to errors are highly desirable to cope with its more fragile nature.

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 [14]. Another relevant part of our ongoing research is devoted to test SPDY over a real satellite Internet Service Provider (ISP).

Acknowledgement. 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] A. H. H. Ngu, M. P. Carlson, Q. Z. Sheng, P. Hye-Young, "Semantic-Based Mashup of Composite Applications", IEEE Transactions on Services Computing, vol. 3, no. 1, pp. 2 – 15, Jan.- March 2010.

[2] L. Caviglione, "Extending HTTP Models to Web 2.0 Applications: The Case of Social Networks", in Proceedings of the Fourth IEEE International Conference on Utility and Cloud Computing (UCC), pp. 361 – 365, Melbourne, Australia, Dec. 2011.

[3] L. Caviglione, "Can Satellites Face Trends? The Case of Web 2.0", International Workshop on Satellite and Space Communications (IWSSC'09), pp. 446 – 450, Siena, Italy, Sept. 2009.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, "RFC 2616, Hypertext Transfer Protocol – HTTP/1.1", Network Working Group, Jun 1999, [onlne]: `http://tools.ietf.org/search/rfc2616`. Last Accessed: Jan 2014.

[5] M. Belshe, R. Peon, "SPDY Protocol", draft-mbelshe-httpbis-spdy-00, Network Working Group, Feb. 2012, [online]: `http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00`. Last Accessed: Jan 2014.

[6] J. Erman, V. Gopalakrishnan, R. Jana, K. Ramakrishnan, "Towards a SPDY'ier Mobile Web", in Proceedings of the ninth ACM conference on Emerging networking experiments and technologies, ACM, pp. 303 – 314, Dec. 2013.

[7] H. Kim, G. Yi, H. Lim, J. Lee, B. Bae, S. Lee, "Performance Analysis of SPDY Protocol in Wired

and Mobile Networks", In Ubiquitous Information Technologies and Applications, pp. 199 – 206, Springer Berlin Heidelberg, Jan. 2014.

[8] G. Mineki, S. Uemura, T, Hasegawa, "SPDY accelerator for improving Web access speed", in Proceedings of the 5th International Conference on Advanced Communication Technology (ICACT), pp. 540 – 544, Jan. 2013.

[9] L. Caviglione, A. Cardaci, A. Gotta, N. Tonellotto, "Performance Evaluation of SPDY over High Latency Satellite Channels", 5th International Conference on Personal Satellite Services (PSATS), Toulouse, France, June 2013.

[10] L. Caviglione, A. Cardaci, N. Celandroni, F. Davoli, E. Ferro, A. Gotta, "SPDY – a new Paradigm in Web Technologies: Performance Evaluation with a Satellite Access Network", 19th Ka and Broadband Communications, Navigation and Earth Observation Conference, Florence, Italy, Oct. 2013.

[11] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, C. Lilley, "Network performance effects of HTTP/1.1, CSS1, and PNG", SIGCOMM Computer Communications Review, No. 27, pp. 155 – 156, 1997.

[12] M. Welsh, B. Greenstein, M. Piatek, "SPDY performance on mobile networks", [online]: `https://developers.google.com/speed/articles/spdy-for-mobile`. Last Accessed: Jan 2014.

[13] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, N. Sutin, "An argument for increasing TCP?s initial congestion window", ACM SIGCOMM Computer Communications Review, Vol. 40, pp. 27 – 33, 2010.

[14] A. Gotta, F. Potorti, R. Secchi, "An analysis of tcp startup over an experimental DVB-RCS platform", in Proceedings of the 2006 International Workshop on Satellite and Space Communications, pp. 176 – 180, 2006.

[15] P. Barsocchi, A. A. Bertossi, M. C. Pinotti, F. Potorti, "Allocating data for broadcasting over wireless channels subject to transmission errors", Wireless Networks, No. 16, pp. 355 – 365, 2010.