Resource Optimization over DVB-RCS satellite links through the use of SPDY

A. Abdel Salam, M. Luglio, C. Roseti, F. Zampognaro
Department of Electronics Engineering
University of Rome "Tor Vergata"
Rome, ITALY
luglio@uniroma2.it, {abdel.salam, roseti, zampognaro}@ing.uniroma2.it

***Abstract* — SPDY is a recent Web technology aimed to improve webpages load time and resource utilization, through the implementation of: multiplexing and prioritization of webpage objects, header compression, resource pushing and encryption. Most of these techniques are already implemented in Performance Enhancing Proxies (PEPs), which are adopted in geostationary satellite systems to optimize TCP-based application performance. Thus, SPDY is expected to provide benefits for satellite communications resource management. This paper provides a careful assessment of SPDY performance over satellite links, compliant to DVB-RCS standard, with return link resources assigned on demand through a Demand Assignment Multiple Access (DAMA) method. Such a reference scenario constitutes a challenging communication environment due to both the limited return link bandwidth and the relatively high latency. Performance evaluation has been carried out through a satellite network emulator, which reproduces physical layer satellite impairments, while running real implementations of both TCP/IP stack and SPDY.**

Keywords—SPDY; HTTP/2.0; Satellite; DAMA; DVB-RCS; PEP; Bandwidth on Demand;

I. INTRODUCTION

SPDY [1] is an open standard developed by Google and currently available in a wide range of web browsers, including Chromium (Google Chrome), Mozilla Firefox and Opera [2]. SPDY has been designed with the aim to reduce webpage load latency using a combination of techniques, including:

1) multiplexing and prioritizing the webpage subresources to be transferred over a single connection per client;
2) header compression using gzip [3] or DEFLATE [4];
3) the use of a server *push* method to proactively send several webpage contents instead of waiting for individual requests from the client;
4) the improvement of security using TLS encryption.

The first draft of HTTP/2.0 standard [5] indicates SPDY as the new Internet application-layer that defines a new semantic to reduce the webpage download time. Thus, its adoption will be widespread in the near future. SPDY is already available to access important Web servers such as Google services, Twitter and Facebook.

Satellite networks can provide a bi-directional link for Internet access. Data transmission usually adopt a DVB-S(S2) broadcast forward link and an interactive return link shared among a set of Return Channel Satellite Terminals (RCSTs), competing for available resources. DVB-RCS standard [6][7] defines several Demand Assignment Multiple Access (DAMA) algorithms in order to efficiently share return link resources, available as MF-TDMA slots. Then, RCSTs can transmit data only on the assigned frequency-time timeslots. In general, DAMA algorithms introduce an extra delay contribution, namely "access delay", which in addition to satellite propagation delay results on a very large delay experienced by TCP-based applications (i.e. HTTP). For this reason, efficiency in the resource utilization is paid with poor performance experienced by users. Under these conditions, adoption of Performance Enhancing Proxies (PEPs) at the edges of satellite link is a usual solution to achieve better performance. PEPs basically rely on TCP splitting and a consequent adoption of TCP enhancements tailored for the satellite segment [7]. Several TCP enhancements, as for instance flow multiplexing, header compression and pre-fetching, present some similarities with SPDY techniques. On the other hand, PEP introduces some incompatibilities with IPsec due to the violation of the end-to-end semantic, while SPDY follows an end-to-end approach enhancing overall security through the application of TLS encryption. Differently from PEP-based architectures, SPDY does not require network modifications, making satellite link an attractive and cost-effective alternative for Internet access in several conditions (i.e. absence of terrestrial access networks, high traffic congestions, high mobility, etc.). Internet providers can then include satellite in the pool of access networks for a transparent multi-segment connectivity. This is breakthrough for satellite role in the future Internet, while in the past it has represented a *weak point* from performance point of view, requiring some protocol and architectural adaptations. In this context, several challenges must be faced up. Satellite physical characteristics are completely different from terrestrial ones, where SPDY efficiency has been largely proven. In particular, the Round-Trip Time (RTT) is about half second with geostationary satellites (herein considered) and return-link resources can consistently and quickly vary due to DAMA. Literature reports as these conditions affect performance of traditional TCP-based applications [7]. Therefore, such physical characteristics represent the main challenge for the efficient application of SPDY over satellite. In case of good performance in comparison with other TCP-based protocol configurations, SPDY approach may represent in the future a successful replacement of PEPs.

In this paper the analysis of the new SPDY Web technology in a satellite link is performed to evaluate if and how much the efficiency already demonstrated over terrestrial networks [2] is confirmed. Typical satellite-specific

978-3-901882-63-0/2014 - Copyright is with IFIP

impairments must be taken into account to verify the protocol applicability in this environment: large latency, additional variable access delay due to DAMA and possible bandwidth limitations. The assessment of resource optimization is achieved through the analysis of SPDY performance over DVB-RCS-like satellite link using an emulated satellite platform. In particular, the suitability of various DAMA allocation mechanisms will be evaluated in several scenarios with different SPDY server configurations.

II. SPDY PROTOCOL

SPDY has been developed and fostered by Google [2], and a related Developer Group has provided the documentation, software implementation and a preliminary performance evaluation. A paradigmatic example is the Chromium Project that has spawned the "*Let's make the Web faster*" initiative [8], with the goal of evaluating SPDY in real-world terrestrial scenarios.

The main SPDY objective is to cope with the increasing traffic coming from Web 2.0 applications. On wired links, SPDY leads a reduction of Web download times in the range of 27-60% [2].

SPDY introduces a framing layer (equivalent to session layer) for multiplexing concurrent streams atop a single persistent TCP connection (or any reliable transport service). As a final result, SPDY aims to transform multiple short transfers into a single longer transfer. Improvements are expected since TCP was originally designed for long bulk transfers. Furthermore, SPDY is optimized for HTTP-like request-response conversations, and also guarantees that all the applications using HTTP can be migrated to the use of SPDY with little or no changes. In more details, SPDY offers four major additional features to outperform traditional HTTP [1]:

- *Multiplexed requests* - there is no limit to the number of concurrent requests that can be sent over a single SPDY connection;
- *Prioritized requests* - clients can request certain resources to be delivered first, avoiding the congestion of the network path with non-critical resources when a high-priority request is pending;
- *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 to complete, the benefit in terms of data reduction could be meaningful;
- *Server pushed streams* - server *push* enables content to be proactively sent from servers to clients without specific requests.

SPDY has a streamlined approach comparing to the traditional transport method adopted by HTTP. The basic idea behind SPDY design is multiplexing different resources over a single TCP connection between the client and the server. Either client or server can create streams independently, with the support of specific control frames interleaved with data frames. SPDY divides the resources data into frames that are streamed in a sequence of independent bi-directional streams.

All such aspects potentially improve performance also on satellite links, allowing a better channel utilization (long TCP connections can achieve optimal throughput even in high delay-band-product links).

III. SATELLITE BANDWIDTH ON DEMAND (BOD):

Access to real time web 2.0 interactive applications with strict requirements in terms of bandwidth, jitter and losses is very challenging in a satellite environment. In fact, geostationary satellite systems introduce a large propagation delay and most of the interactive links adopt DAMA allocation schemes to efficiently share return link resources. DAMA introduces a variable delay contribution, called "*access delay*", to the overall delay experienced by applications. With reference to DVB-RCS standard, DAMA algorithms rely on a control loop where specific control messages for the request of resources are exchanged between a Network Control Centre (NCC) and the RCSTs.

In particular, DVB-RCS standard defines three main types (out of five available) of DAMA allocation algorithms, which can also be used in combination [6]:

- *CRA* (Continuous Rate Assignment) is a fixed allocation, with some transmission timeslots permanently allocated to an RCST irrespective of whether they are actually used or needed; there is no allocation delay experienced, while resources utilization can be very inefficient;
- *VBDC* (Volume Based Dynamic Capacity) is a dynamic allocation based on requests of the RCSTs related to the volume of data stored in the MAC buffer; each request is associated to a given volume of traffic, and requests are cumulative; VBDC is typically characterized by large access delay and high resources utilization;
- *RBDC* (Rate Based Dynamic Capacity) is a dynamic allocation; requests are issued for a given data rate (i.e., rate at which data feed MAC buffer) and are valid for a certain time; RBDC introduces an access delay, as for resources utilization, in between CRA and VBDC cases.

IV. TEST CAMPAIGN

Test objectives are to provide an assessment of SPDY performance over a geostationary satellite link with different DVB-RCS DAMA algorithms running at the link layer. In addition, performance of SPDY protocol will be compared with the one of the traditional HTTP protocol. Then, tests will aim to highlight how the overall variable delay impact on the web request-response loops using real protocol reference implementations.

Reference scenario envisages a satellite star-based architecture, where a pool of satellite terminals shares satellite return link in compliancy to DVB-RCS standard (Section III). Each satellite terminal sends data over satellite channel to the satellite gateway, which provides interfaces to Internet. In the following tests, remote web servers are assumed directly connected to the satellite gateway, since both delay contribution and bandwidth constraints of the transport networks can be considered negligible with respect to satellite values.

A. *Testbed description*

SNEP [9] is a satellite Virtual Network emulating the behavior of components of a real DVB-RCS network. The virtual emulation environment is managed by the VMware vSphere Hypervisor ESXi software [10]. SNEP reproduces satellite network communication environment (i.e. delay, bandwidth, TDMA framing) as well as DAMA algorithms for the dynamic capacity assignment on the return link. The core components of SNEP are:

- The Gateway (access router);
- The Network Control Centre (NCC) or Hub;
- The Satellite;
- The RCS Satellite Terminals (RCST);
- The User Terminals (UTs), connected to the RCSTs through a Local Area Network (LAN).

UTs running both SPDY implementation and real web applications can be integrated with the platform at any point. Accordingly, a web browser has been connected to a RCST (user side), and a SPDY-enabled web server to the satellite gateway at the other end of the satellite link.

Both external web client and web server rely on quad-core CPU servers with a 32 gigabytes RAM. Therefore, hardware performance is not a limiting constraint in any of the performed tests. In Table I the detailed software configuration of client and server end points is reported.

B. *Test Setup*

Test setup envisages three main element: a web client installed on a UT (connected to a target ST), the SNEP core reproducing a DVB-RCS satellite and a web server installed on a terminal behind the emulated satellite gateway.

TABLE I. WEB SERVER AND CLIENT CONFIGURATION

Parameter	Value
Operating System:	• Ubuntu 12.04 LTS/ i686. Linux kernel 3.2
Web Browser	• Google Chrome 26.0.1410.40
Benchmarking and Analysis tools	Browser equipped with the following debug tools • Page Benchmarking tools for Chrome [11]; • Speed Tracker for Chrome [12]; • SPDY Indicator [13]; • Google Chrome Developer Tool (GCDT) [14], • SPDYShark (SPDY module for Wireshark) [15].
Web Server configuration	• Apache/2.2.22 [16] • SPDY/3.0: Mod_SPDY 0.9.4.1 [17] • Server Push (X-Associated-Content header) enabled with different percentage for the pushed objects [17]. • Apache KeepAlive settings enabled / disabled All other default Apache configurations are considered
Test web page	HTML web page consists of 640 small images (index.html 24.8 kbytes and 640 small images totalling 333 kbytes)

The considered RCST can alternatively enable one of the BoD DAMA algorithms described in Section III. CRA (Constant Rate Assignment) is the basic scheme: all the time-slots are permanently assigned to the RCST for the whole simulation duration. In alternative, either Rate-Based (RBDC) or Volume-Based (VBDC) algorithm can be enabled to dynamically assign return link time slots, depending on the traffic outgoing from RCST.

The web server is configured to alternatively support SPDY and traditional HTTP under different configurations (i.e. HTTP/1.0, HTTP/1.0 over SSL, HTTP/1.1 with Pipelining). The complete platform configuration and protocol parameters are specified in the Table II according to the specific test goals.

TABLE II. TESTBED SETUP

Parameter	Value
Bandwidth	• 4/1 Mbit/s FWD/ RTRN
BoD DAMA Settings	• CRA (228 slots as a constant rate) • RBDC (228 rate based slots) • VBDC (228 volume based slots) • Mixed DAMA configuration (18 CRA slots, 105 RBDC slots, 105 VBDC slots)
Web protocols	• SPDY/3.0 • HTTP1.0 • HTTP/1.0 + SSL (self-signed certificate) • HTTP/1.1 + Pipelining
Server Push	• Enabled with different percentage of the pushed objects.
TCP configuration	• TCP Cubic (configurable TCP parameter will be selected to optimise performance over satellite) • tcp_moderate_rcvbuf ON (dynamic TCP buffer) • Initial Window = 10 (boost connection startup)

A static HTML web page has been implemented for test purposes. In order to stress web request-response iterations, the webpage is structured as an *index.html* of 24.8 Kbytes and 640 small images. This page is built specifically to highlight its transfer time over a high RTT links and to stress the various innovations brought by SPDY: object push, header compression and multiplexing.

C. *Methodology*

PEPs are network elements that terminate TCP connections at the edges of a satellite link in order to perform a protocol conversion with the aim to improve overall performance. Actual PEP implementation does not follow any specific standard, often involving ad-hoc protocols. For this reason, a PEP reference configuration for test purposes is not easily identifiable. As a preliminary analysis step, work performed in this paper focuses on an assessment of SPDY performance over satellite, compared to other Web end-to-end configurations. The goal is to evaluate whether or not SPDY performance improvement is relevant and overall results are close to those experienced on terrestrial-only links. Considering test Web page configuration (Table I), load time measurements have been achieved performing downloads over an ADSL link. Total load time is about 15 s in average, while first paint time is lower than 1 s. There values will be used as benchmark. In case of positive outcomes, a detailed comparison with PEP implementations should became not necessary, considering PEP drawbacks in terms of security and required network modifications.

The Page Benchmarking tool [11], installed on Google Chrome browser (user terminal side), has been used to access the testing webpage and view values of interest. Benchmark tool is configured to close the connection and clear the browser

cache before each test iteration to ensure that all page objects are requested and downloaded from the server from scratch. In fact, 50 iterations run for each configuration and results are averaged to minimize variability of the network.

For a fair comparison, tests related to traditional HTTP are replicated also enabling both pipelining option in the client browser and KeepAlive option in the apache server, to enable HTTP – Pipelining on TCP persistent connection [18]. This setup makes HTTP protocol more similar to SPDY reusing the same connection for multiple requests/responses. Again, page-benchmarking tool is used for the request of the testing webpage using HTTP1.1 pipelining.

All the above tests have been repeated over different BoD DAMA algorithms as indicated in the test setup.

V. RESULTS

Fig. 1 shows test results in terms of page load time (PLT), meant as the time between the first HTTP request and the closing of all the opened connections. In addition, downloaded objects can be displayed as soon as they are received, without waiting for the download of the whole page, as described in a successive test. As a general simulation outcome, SPDY outperforms all the other HTTP configurations, and the performance improvement increases at higher RTTs (experienced in case of DAMA VBDC and RBDC algorithms). For instance, SPDY reduces page load time by 92% comparing to HTTP and by 85% comparing to HTTP with pipelining/keep alive over VBDC. SPDY absolute PLT ranges from 22.7 s in VBDC to 9 s with CRA. This demonstrates how SPDY allows performance very close to that experienced over typical terrestrial networks (15 s if adopting baseline HTTP/1.0). Performance of HTTP/1.0 without persistent connection (dashed bars in Fig. 1) are very poor over all the DAMA schemes. The primary reason is that the browser opens a new TCP connection to fetch each object (total of 642 connections). Management of a high number of short connections is time consuming, since each connection needs an initial three-way handshaking (1 RTT spent for signaling), TCP performs short data transfers in the Slow Start phase (low data rate) and an additional RTT is spent to close the connection. With SSL on, HTTPS performance is further impaired. SSL doubles the number of needed connections (1290 connections), because with a self-signed certificate Chrome browser adds a SSL handshaking before each object request. Enabling HTTP/1.1 by setting the KeepAlive option ON in Apache configuration and allowing Chrome to use pipelining enhances performance by sending multiple requests from the client over the same socket without waiting for the server response. In particular, HTTP/1.1 opens only 12 connections for page download, instead of 642 without pipelining, since Chrome starts 6 parallel connections by default, every connection can be reused for 100 request (default Apache configuration). Finally, SPDY reduced the number of connections to only 4 for the full-page download: one connection used to transfer the webpage objects, the others created by Chrome browser as backup and speculative [19] ones. The lowest number of connections combined to object multiplexing allows a great performance improvement with SPDY. Overall PLT is drastically reduced and it is only marginally impacted by the RTT increase due to BoD DAMA. Achieved values, ranging from 3 to 25 s, can be considered acceptable for end user experience. Therefore, this first result collection demonstrates effectiveness of the SPDY approach to manage Web transfers, making it a valid alternative to the traditional PEP approach.

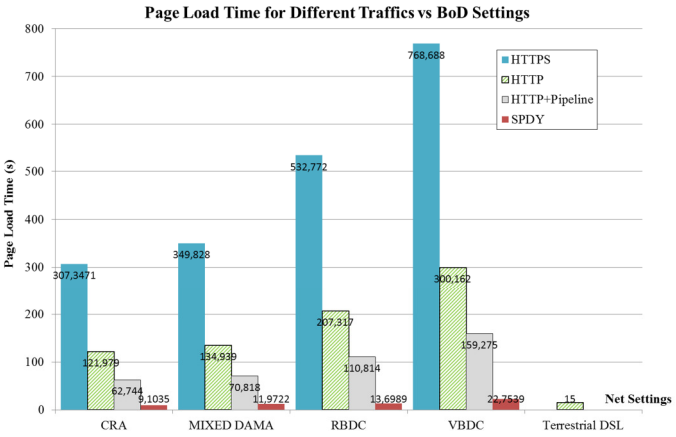

Fig. 1. Overall page load time evaluation

Fig. 2 provides a focus on the connections management for the considered configurations. As seen above, the number of connections is an important metric to foresee overall performance. A higher number of connections has two main drawbacks:

1- a higher TCP signaling exchange leading to a relevant delay contribution experienced by users;

2- each TCP connection is used to carry a single small object; as a consequence, TCP performs the transfer in the Slow Start phase at a low data rate.

Results show how SPDY takes advantage from *multiplexing* technique to manage the transfer over a single connection, allowing TCP to increase transmission rate up to the optimal value (equal to the available bandwidth), thanks to the Congestion Avoidance algorithm. HTTP/1.1 with pipelining enabled needs of a threefold number of connections , while HTTP/1.0 with/ without SSL opens a number of connections much higher.

Another feature included in SPDY is *header compression*. It allows reducing the amount of effective transferred bytes. Of course, compression effects are more evident in case of large amount of transferred data. In the considered scenario, where a single webpage download is analyzed, header compression does not have much impact, but it still provides some benefits to overall performance.

As shown in Fig. 3 with HTTP page download consumes 508 Kbytes, including all packet-level headers. With SPDY, the number of bytes consumed is only 416 Kbytes total, although encryption is performed. For comparison, adding encryption on HTTP (HTTPS), the browser receives about 870 Kbytes due to the encryption and security handshaking.

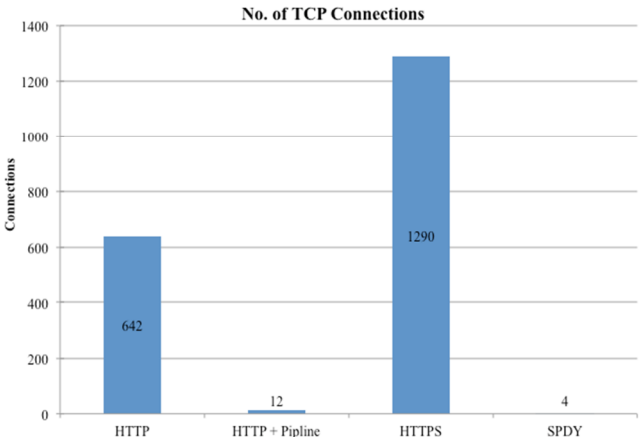

Fig. 2. Monitoring of connections

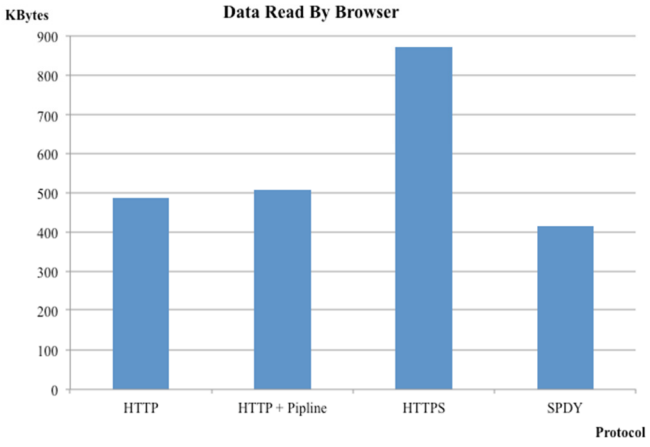

Fig. 3. Data read by browser

To assess more in depth user experience, the "Time of the first paint" available within the benchmarking tools can be evaluated. This parameter indicates the time when browser starts to render the web page. As a user-level qualitative metric, this parameter is more representative than the overall page load time, since it is a better indication of content readiness (first objects starts to appear on the browser window). Results are shown in Fig 4. In general, HTTP starts rendering the page faster than SPDY, since the latter needs an additional time for object multiplexing, header compression, and framing process. From result analysis, it is possible to conclude that the rendering start is constrained by security-related operations, but it is still lower than the HTTPS case. Therefore, with SPDY the time of the first paint is very close to time needed to perform the overall transfer. From a different perspective, joining results in the Fig 4 with those in Fig.1, it is possible to state that with SPDY the time to complete a web page transfer is very similar to time to the first paint with HTTP.

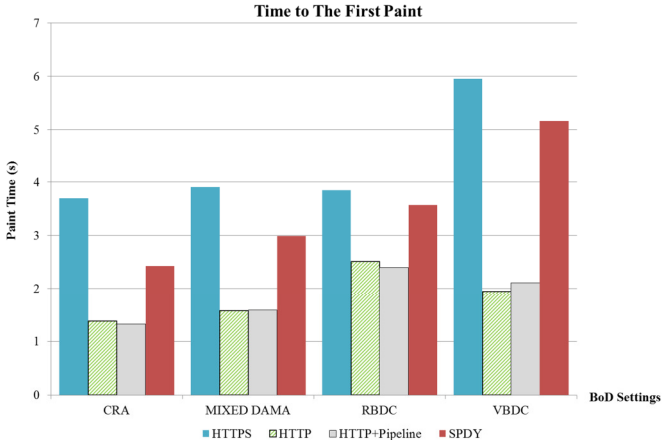

Fig. 4. Time of the first paint

A last important feature introduced by SPDY is *server push* [1]. Pushing method consists on proactively transmitting a number of objects without waiting for the corresponding requests from the client. In this way, the PLT can be reduced by eliminating the round trip time between a client request for a resource and the corresponding server's response, while using at best available bandwidth. Server push configuration is usually set with the scope to send at once all the resources believed important for the client. However, SPDY basic mechanisms already envisage multiplexing with priority of multiple objects on a single connection. Therefore, expected advantages are not significant. Several tests have considered SPDY with different percentage of pushed objects over different BoD settings. This feature in HTTP(S) is not available. Results in Fig. 5 show that SPDY download time is quite independent from the percentage of pushed objects. Again, overall performance depends on the experienced RTT, so that CRA outperforms the other schemes since leading to an RTT close to the two-way propagation delay only. On the other hand, VBDC provides the worst performance due to an overall RTT of about 1.6 s. While the page load time is not particularly impacted by server push, the time required by the browser to start rendering (defined as time to the first paint) increases proportionally to the percentage of the pushed objects. This is due to resource encoding and header processing required by pushing process. The increase of time to the first paint as a function of an increasing percentage of the pushed objects is shown in Fig. 6

After a characterization of performance experienced by users, the last simulation tests focus on network-related performance in terms of resource utilization. The overall transmission rate has been first monitored on both forward (web page download) and return (web requests and signaling) links. In addition, time-slots actually allocated on the return link have been tracked for the different application protocol configurations. Without lost of generality, the mixed DAMA algorithm, as presented in the Table II, is considered for simulations.

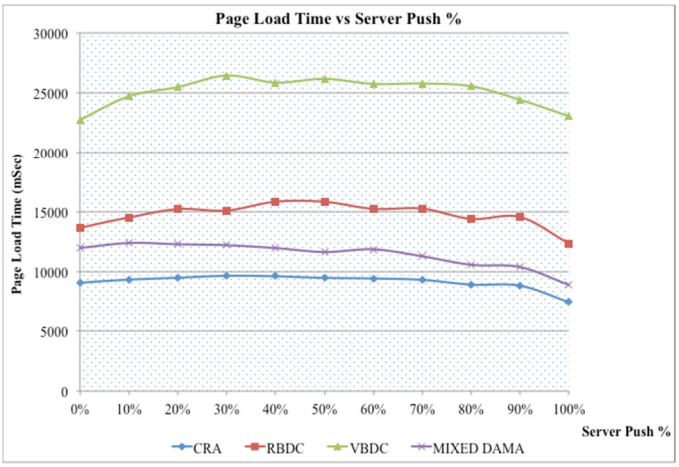

Fig. 5. SPDY Page load time with % server push

Fig. 7 reports throughput (kbit/s) on both satellite downlink and uplink. SPDY bandwidth profile presents a spike in the downlink corresponding to the download of all objects multiplexed on a single connection and sent at once. This behavior reduces also traffic measured on the return link, thanks to the limited number of active connections and the low number of requests generated by the client. Then, download terminates in few seconds as already proved in the previous simulation tests.

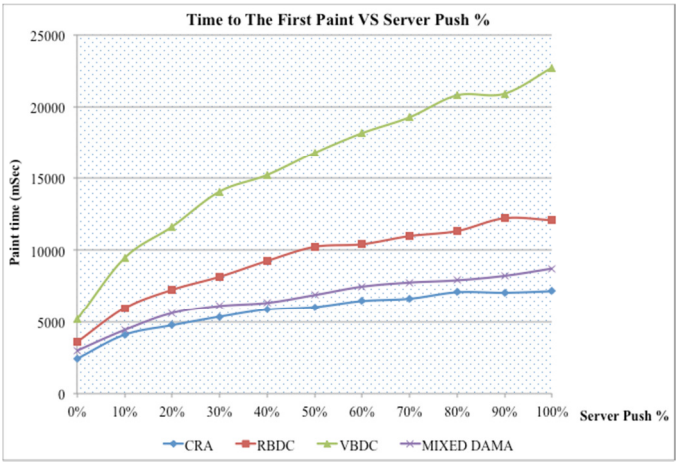

Fig. 6. SPDY first paint time with % server push

HTTP/1.1 with pipelining leads to a web download spread over a longer interval time. Although pipeline allows to send multiple object requests together, object downloads are still sequentially sent, limiting the maximum achievable rate over the TCP connection (application-limited rate). In addition, signaling over the return link is increased: it is about one half of the traffic over the forward link.

HTTP/1.0 (no pipelining) presents even worse performance in terms of both achieved data rate and traffic over the return link:

- Download is performed at about 50 kbit/s out of the 4 Mbit/s available;
- Traffic on the return link is very close to the amount of data downloaded; then, a higher request of shared return link resources will be needed.

As usual, HTTPS presents the worst performance. Even SSL signaling leads return link traffic to exceed the amount of downloaded data. This result demonstrates that the lower PLT is in particular associated to a much better exploitation of the available bandwidth if using SPDY.

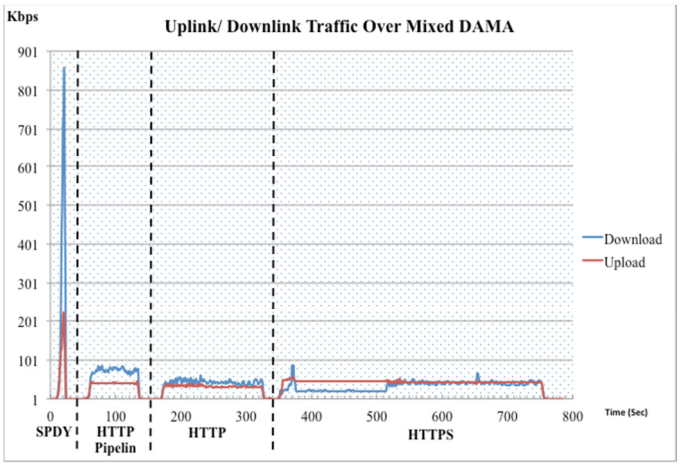

Fig. 7. Bandwidth utilization during web page download

Finally, Fig. 8 shows the dynamic assignment of available slots on the return link (using the mixed DAMA BoD) when web page download is performed four consecutive times varying the configuration. SPDY allows a better utilization of the resources getting a large number of slots for a limited time interval: at around 15 s all the available slots are used to send most of data. The overall oscillatory trend is due to the burst nature of the TCP transmission on links with large delay-bandwidth product [7]. In general, simulation outcomes show how SPDY transmission based on multiplexed objects allows a rapid allocation of the needed resources, limiting the impact of the DAMA allocation control loop.

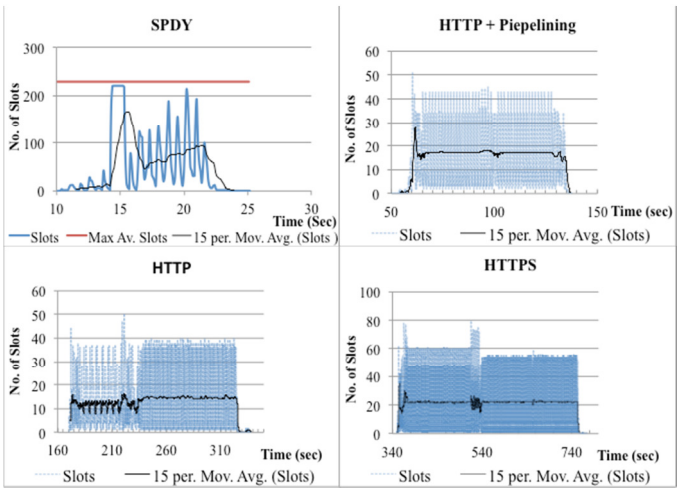

Fig. 8. Return Link Slots

To opposite, with the other configurations, overall transfer is segmented into sequential smaller data chunks (i.e. a single web object) on different TCP connections. As a consequence,

when a new connection is established, slots are requested proportionally to the carried data (a smaller amount than in the SPDY case). Then, when connection terminates, previous assigned slots are released and requested again when a new connection is established. Definitively, the average number of slots is much lower (black lines in the Fig. 8), while the overall transfers last a longer time period.

VI. CONCLUSION

This paper presents the improvements achievable when using SPDY over a satellite link, adopting DVB-RCS DAMA on the return link. An ah-hoc test Web page including many objects has been specifically configured on a Web Server and downloaded from a test client, connected to an emulated satellite terminal. Analysis has taken into account the main mechanisms introduced by SPDY over different DAMA configuration. Test results confirmed a significant performance improvement in all target configurations. In particular, most of the measurements show how SPDY allows a PLT very close to measurements achievable on terrestrial-only links.

The slightly higher first rendering time (first paint) is well compensated by a lower overhead (less connections and bytes transferred) and a better network efficiency (lower download time and higher achieved throughput). Furthermore SPDY introduces the novel push mechanism, which can be exploited to enhance user experience even more, sending at once higher priority content. Performance improvement showed herein makes SPDY as a valid solution to accelerate Web applications over satellite, allowing to reconsider the role and the need of PEP agents as unique viable solution to achieve acceptable Web performance over satellite.

ACKNOWLEDGMENT

This work is performed as part of the European Space Agency (ESA) Satellite Communications Networks of Excellence (SatNEx) Network of Experts, Phase III.

REFERENCES

[1] Belshe, M., Peon, R.: SPDY protocol - draft 3. http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3 (2012)

[2] The Chromium Projects, “SPDY: An experimental protocol for a faster web”, http://www.chromium.org/spdy/spdy-whitepaper

[3] P. Deutsch, “GZIP file format specification version 4.3”, RFC 1952, May 1996

[4] P. Deutsch, “DEFLATE Compressed Data Format Specification version 1.3”, RFC 1951, May 1996

[5] M. Belshe, Twist, R. Peon, M. Thomson, A. Melnikov, “Hypertext Transfer Protocol version 2.0”, IETF, draft-ietf-httpbis-http2-01, (Jan. 2013).

[6] Digital Video Broadcasting (DVB); Second Generation DVB Interactive Satellite System (DVB-RCS2); Part 2: Lower Layers for Satellite standard”, DVB Document A155-2, Jan 2013.

[7] M. Luglio, C. Roseti, F. Zampognaro, Performance evaluation of TCP-based applications over DVB-RCS DAMA schemes, International Journal of Satellite Communications and Networking, Volume 27 Issue 3, Pages 163 – 191, Published Online: 2 Mar 2009, DOI:10.1002/sat.930

[8] Google Developer Webpage, “Make the Web Faster”, https://developers.google.com/speed

[9] M. Luglio, C. Roseti, F. Zampognaro and F. Belli, An Emulation platform for IP-based satellite networks, 27t International Communication Satellite Systems, Conference (ICSSC 2009), 1 – 4 June 2009, Edinburgh, UK

[10] Wmware, Wmware vSphere Hypervisor, http://www.vmware.com/ products/ vsphere-hypervisor/overview.html, US

[11] The Cromium projects, Benchmarking Extension for Google Chrome https://sites.google.com/a/chromium.org/dev/developers/design-documents/extensions/how-the-extension-system-works/chrome-benchmarking-extension

[12] Google Web Toolkit “Speed Tracker”, http://developers.google.com/web-toolkit/speedtracer

[13] SPDY Indicator for Chrome, https://chrome.google.com/webstore/detail/spdy-indicator/mpbpobfflnpcgagjijhmgnchggcjblin?hl=en

[14] Chrome Dev Tools, https://developers.google.com/chrome-developer-tools

[15] Wireshark Plugins for SPDY, https://code.google.com/p/spdyshark/wiki/BriefInstallInstructions

[16] Apache, the HTTP Server Project, http://httpd.apache.org/, latest release Jul 2013.

[17] Google, “Mod_spdy, open-source Apache module”, http://developers.google.com/speed/spdy/mod_spdy/, May 2012

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

[19] William Chan, Connection Management in Chromium, Feb 2013, https://insouciant.org/tech/connection-management-in-chromium/