Understanding Multistreaming for Web Traffic: An Experimental Study

M. Rajiullah[†], A. C. Mohideen[⋆], F. Weinrank[‡], R. Secchi[⋆], G. Fairhurst[⋆] and A. Brunstrom[†]
[†]Karlstad University, Karlstad, Sweden
{mohammad.rajiullah, anna.brunstrom}@kau.se
[⋆]University of Aberdeen, Aberdeen, U.K.
{althaff, raffaello, gorry}@erg.abdn.ac.uk
[‡]FHM, Munster, germany
weinrank@fh-muenster.de

***Abstract*—This paper explores the design trade-offs needed for an Internet transport protocol to effectively support web access over HTTP/1.1. It explores use of a multistreaming transport protocol mechanism and studies these using a practical methodology utilising the transport features of TCP and SCTP. This is used to evaluate the relative benefit of key transport mechanisms and analyse how these impact web access performance. Our conclusions help identify the root causes of performance impairments and suggest appropriate choices when selecting a suitable transport protocol.**

***Keywords*—TCP; SCTP; Web; performance analysis**

I. Introduction

Over the last decade web pages have evolved from a simple compositions of images and text, to highly complex structures embedding interactive contents and web applications [1]. A typical web page is a collection of inter-dependent resources whose order of delivery and processing impacts the page display time. Recent studies [2]–[4] found that the dependency graph for web page resources (and corresponding scheduling order) play a significant role in overall web performance.

A problem known as head of line blocking (HoLB) occurs when the chain of processing is delayed waiting for a critical resource to be received over a transport connection [5]. Browser/server implementations employ various techniques to mitigate this and thereby accelerate page download [5]. One approach is to increase the *parallelism* of resource download, i.e., to request/retrieve an HTTP resource while other resources are being downloaded. HTTP/1.1 [6] allowed a client to open two parallel TCP connections that may be kept open for multiple request/response transactions (known as HTTP persistence). Since early specifications of HTTP/1.1, browsers have used an even larger number of connections per server (e.g., the current default is six in Mozilla Firefox and Google Chrome) and often adopted a proactive policy of connection management, including closing/reopening slow TCP connections or requesting the same resource over multiple connections. In addition, servers often distribute web pages across multiple domains (even for the same server), a practice known as *sharding*. This forces a client to open multiple connections, at least one for each domain [7]. This contributes to further increase the required number of transport connections.

ISBN 978-3-901882-94-4

Although parallelism has benefits, introducing a large number of TCP connections is not without drawbacks. First, the client-server session may experience a large number of under-utilised connections (e.g., a TCP connection to send only a small resource), which reduces efficiency due to the overhead required to open and maintain each connection. Second, breaking the transmission flow into many independent TCP connections reduces control over congestion, making web traffic more aggressive towards other competing traffic [8].

There are drawbacks in using multiple parallel connections to the same web server, but this is still commonly used by web clients [9]. On the one hand, applications open parallel connections to achieve concurrent object transmissions because TCP does not offer a mechanism to identify sub-streams. On the other hand, the scarce deployment of message-oriented protocols, such the Stream Control Transmission Protocol (SCTP) [10] [11], means there is little viable alternative.

This paper evaluates the benefits of multistreaming extending the analysis in [11]. The contribution is two fold: First, a web traffic workload based on both a dependency graph and the processing time for HTTP objects at a web client is used to explore the benefits of multistreaming. Second, the evaluation considers the impact of RTT and capacity on the web performance.

The remainder of this paper is organised as follows: Section II describes our web model and testing methodology. The experimental tool and experiment are described in section III followed by the performance analysis in section IV. The paper concludes in section V.

II. Web Model and Dataset

This study utilised a public web performance dataset [12]. This provides the number and size of HTTP resources from 170 recorded web pages. It also includes graphs representing the dependency between HTTP resources and their processing time at the client.

To characterise the web traffic workload, the web pages were categorised according to the total size of all resources in a page. This total was used to divide pages into 6 bins (size-ranks), labeled A to F. The bins were organised so that each size-rank held an equal number of web pages, forming statistically significant groups. Table I reports the interval of

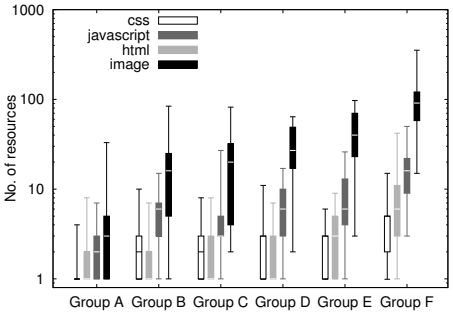


Figure 1: Distribution of number of resources by MIME type across size-ranks.

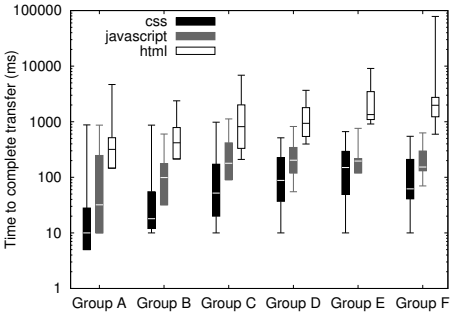


Figure 2: Distribution of time to complete transfer by MIME type across size-ranks.

sizes for each size-rank in the second column, and the 5%, 50% and 95% percentile for the resource size distribution in the 3rd, 4th and 5th column. For each bin, the percentile of the distribution of the number of resources at 5%, 50% and 95% is also reported in parenthesis.

This data shows a correlation between the size of a page and the number of resources. There is a wide distribution in the number of resources within each size-rank. For example, in the smallest size-rank (A) the number of resources/page varied between 1 and 39, whereas the largest size-rank (F) ranged between 49 and 228 resources/page. This suggests that pages of similar size may have a quite dissimilar composition and it may not be sufficient to characterise web pages only by their overall size. However, for simplicity, our experiments consider only the webpage with median size for each size-rank.

The size of the retrieved resources was also observed to be correlated to the total web page size, i.e. larger webpages tend to transport bigger (and often more complex) resources, such as video or interactive banners, and tend to cluster multiple items in a single resource, e.g., using a single javascript file to send multiple scripts. However, the distribution of resource size has less spread than the distribution of the number of resources.

Table I: Webpage size and 5, 50 and 90 percentile of number of resources per size-rank.

Group Name	Size-Rank (KB)	Size (KB) and # res. at 5%	Size (KB) and # res. at 50%	Size (KB) and # res. at 95%
A	0.05-118	0.05 (1)	23 (6)	109 (39)
B	119-565	129 (3)	325 (21)	532 (67)
C	566-873	567 (6)	690 (25)	846 (69)
D	874-1242	878 (6)	964 (45)	1183 (82)
E	1243-1945	1286 (24)	1546 (55)	1901(119)
F	1946-3315	2070 (49)	2454 (127)	3309 (228)

Figure 1 categorises the resources by their MIME type. This result shows the four most common types: text files (*HTML*), scripts (*javascript*), style-sheets (*CSS*) and images. Images were by far the most common, forming the main category in each sizerank. We also observed very few image URLs, suggesting the dependency graph grows mainly horizontally (i.e. increasing number of branches originating from a single resource). Other types include Flash resources, octect-stream and fonts, but these contributed less than 2% of resources.

Figure 2 shows the distribution of the time spent by the client to complete transfer of a resource (including computation time). This figure excludes images because images are terminal nodes in the dependency graph. We observe that the time component was often not negligible in comparison to the transfer time for the resource over a typical network path RTT (few tens of milliseconds). The total time for web pages represented by largest size-ranks (E, F) was around or above one second. This non-negligible latency impacts transport performance and is therefore discussed later in the paper.

III. Tools and Experiment Setup

A. Experimental Testbed

Our performance analysis used a set of three computers emulating a web client, a network, and a web server. All computers had a common hardware configuration of 4 GB RAM and Intel Core 2 Duo processor (2.6 GHz). The network was emulated by the Dummynet traffic shaper [13], configured with a bottleneck capacity, delay, buffer size, and packet loss rate.

We modelled a range of path RTTs representative of both desktop and mobile users, drawn from a distribution derived from an empirical study at Mozilla, Table III. Both the client and server supported TCP (Linux ver 4.2.0-42 and BSD) and SCTP (under BSD). The multi-streaming web server is described in section III-C. A custom made client emulated a HTTP/1.1 browser (section III-B), enabling requests with either a number of parallel TCP connections or a single SCTP connection using a number of streams. We use different number of concurrent SCTP streams 1, 6, 18 and 100. We assume this is a suitable range to support most web pages. More on cost of opening streams is discussed in Section IV-D. Our experiments used a maximum of 18 parallel connections [1]. The same Initial Window (IW) was used for TCP and SCTP. The client used an IW of three packets, recommended by the IETF and common for windows users. The server used an IW of 10, common for Linux-based servers, and an experimental IETF specification. The maximum segment size was 1500 bytes. The experiment parameters are summarised in Table II.

[1]Common browsers open up to six connection to a single domain, but sharding contents across multiple web servers is also common.

Experiment parameters		
Category	Factor	Range/value
Network	RTTs	20, 50, 100, 200, 800 ms
	Bottleneck Capacity	2, 10, 100 Mbps
	Packet loss	No loss, 1.5%, 3%
TCP/SCTP	IW	client (IW 3), server (IW 10)
	CWND validation	no
	# parallel TCP flows	1, 6, 18
	# streams in SCTP	1, 6, 18, 100
Requests	Cookie Size	NULL, 512 B, 2K

Table II: Experimental parameters

Percentile	Desktop RTTs (ms)	Mobile RTTs (ms)
5	1	11
25	20	44
50	79	94
75	194	184
95	800	913

Table III: Path RTT from data provided by Mozilla

B. pReplay Web client

Web requests were generated using the pReplay tool[2], developed in C. This uses libcurl [14] to replay HTTP traces using HTTP/1.1 over TCP or a modified version of phttpget [15] extended to support SCTP [16]. The tool used a dependency graph in JSON files that represented the resource requests and computation times required to process java scripts, CSS etc. pReplay walks the dependency graph, starting from the first activity to load the root HTML. When a network activity is found, pReplay issues a http request using the relevant URL. For computational activity, the tool optionally waits for a time determined by the graph. Once an activity completes, pReplay checks whether all dependent activities have also completed and then commences the next activity. It finishes only when all activities in a dependency graph have been visited.

C. Lightweight Web Server

The server was a modified lightweight web server *thttpd* (tiny HTTP deamon) [17] supporting HTTP/1.1. This work extended a previous patch that allowed thttpd to be run over SCTP [18]. The original patch only enabled web traffic to use a single stream for each SCTP association. This work extended [18] to enable parallel multi-streaming, with the possibility to introduce algorithms for sharing transmission opportunities between parallel streams (i.e., sender scheduling using a round-robin or another algorithm), and support for interleaving large objects (i.e., SCTP I-DATA [19]).

IV. Results

This section contains a systematic study of web page load time (PLT) using HTTP/1.1 over both TCP and SCTP. Our goal is to understand the conditions that benefit the use of multiple connections compared to multistreaming. pReplay was used to measure PLT the time between making the first web request and the time either the last response is received or the last computation is completed. The results present data for an average of 30 runs, plotted with 95% confidence intervals. Cookies were not found to influence the PLT. We therefore exclude results with different cookie sizes. We only show results for the websites at the 50th percentile from our web model (Section. II). The statistics of the sites are described in Table IV.

The dataset processing time [12], was used as an upper bound to analysing the impact of processing time. Since this data was collected, advances in client platforms and in the way resources are parsed and processed have reduced this bound. We therefore also plot the load times with no additional processing time, to present a minimum bound. Although we explored all cases, our experiments did not observe significant differences among the behaviour for different transport mechanisms for different bottleneck capacity. The relation between protocol performances is similar, we therefore only show results for 10 Mbps capacity.

A. Impact of parallelism

Figure 3 shows the impact of PLT with different numbers of parallel TCP connections compared to a single SCTP connection supporting multiple streams (100 in this case[3]). We first discuss the case of no processing time dependency and no emulated loss (tail drop loss from router buffers was observed in some experiments).

Each transport connection independently performed start-up, congestion control and loss recovery. At any one time, one transport pipe can only send a single web resource. In contrast, parallelism allows multiple transport pipes to each simultaneously send a resource, reducing the PLT when multiple resources can be sent. Parallelism was introduced using parallel TCP connections (each independently managing congestion control) or using multiple SCTP streams (where all streams shared a single congestion controller).

Figure 3 shows the benefit of increased parallelism. In general, the PLT improved, except in Figure 3e, where, six and eighteen connections have a similar PLT. Pages with fewer resources (see Table IV), may be retrieved from a single server, and the parallelism may be limited to less than 6 parallel pipes.

However, parallelism also came with a cost:

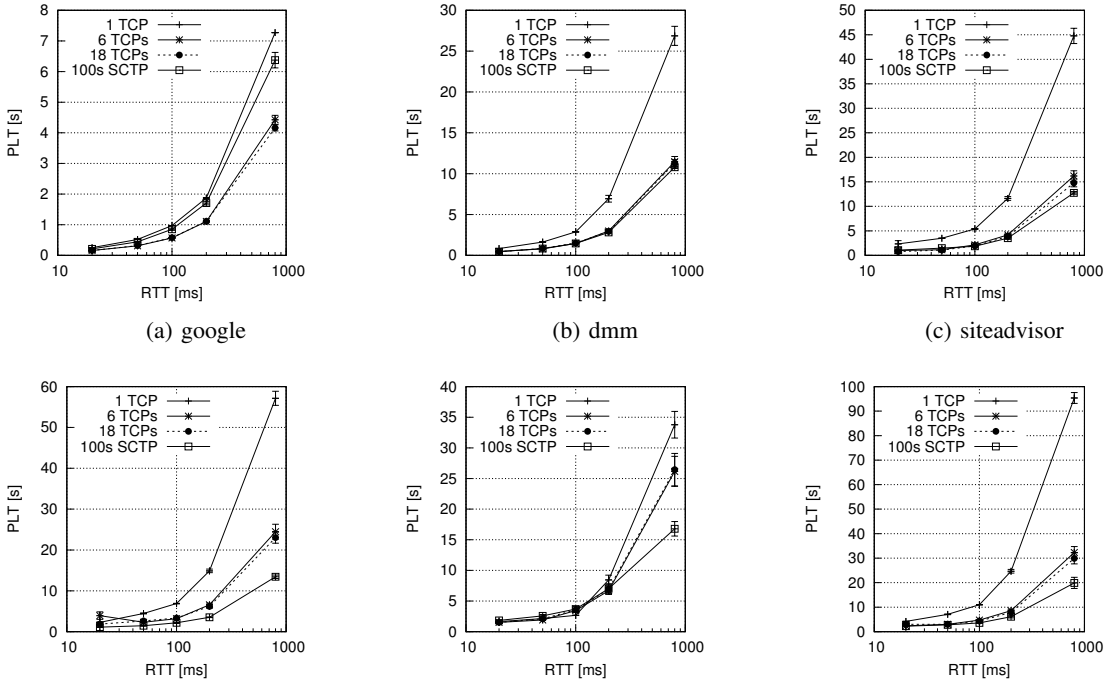
- For a transport protocol with an independently managed congestion control (TCP), a higher sending rate can induce congestion leading to collateral damage to other flows sharing the bottleneck (which we did not consider further in our experiments).
- For a multi-streaming transport protocol using shared congestion control (in this case, SCTP) each stream contributes to the capacity of the association. This increases congestion window growth, reducing the PLT (Figure 3). This also reduces the throughput when congestion is experienced.

[2]Based on Epload [12]

[3]results with other numbers of streams were omitted for better readability.

Table IV: Statistics for the web pages in experiment

Page	Res. Count	Page Size (KB)	≈ Av. Res. Size (KB)
Google	8	74	9
Dmm	21	330	15
Siteadvisor	40	701	17
Amazon	53	977	18
Pinterest	6	1548	258
Mediafire	75	2474	33



(a) google (b) dmm (c) siteadvisor

(d) amazon (e) pinterest (f) mediafire

Figure 3: PLT for 10 Mbps capacity, no loss, without processing time.

This will have the positive effect of reducing collateral damage between streams, i.e. is more friendly to other network traffic.

In most cases, (except for the *google* sites in Figure 3a), a multi-streaming approach provided a smaller PLT than the *N* parallel TCP pipes, which suffer overhead from setting-up multiple connections, and self-induced congestion from concurrency. For small pages, (e.g. *google* in Figure 3a), the combined IW provided by *N* TCP connections provides a benefit over the single IW with multi-streaming.

Figure 3 shows a higher PLT at a larger RTT. However, multi-streaming shows benefit for higher RTT paths, where connection overhead is important. For example, in Figure 3f, when RTT increases from 200 ms to 800 ms, PLT increases over 282% using 18 TCP parallel connections, compared to 229% using multi-streaming.

Web page structure also plays a key role in determining the PLT. When there is no parallelism, the number of resources influences PLTs more than the overall page size. This may be seen in Figure 3d, for 1 TCP, where the *Amazon* page (with a larger number of smaller resources) complete much later than the *Pinterest* page in Figure 3e (with fewer larger resources), (see Table IV). Parallelism alleviates this by reducing the head-of-line dependency delay for pages with many resources. As can be seen, the PLT for *Amazon* is lower than the PLT for *Pinterest* when multi-streaming or *N* parallel TCP connections are used. The larger PLT for *Pinterest* in Figure 3e, is limited by the size of individual resources for the high RTT scenario (800 ms), where additional parallelism can offer benefit. Performance using HTTP/1 could be improved in this case by Spriting the images (dividing a single image into multiple resource files).

B. Impact of processing time at the client

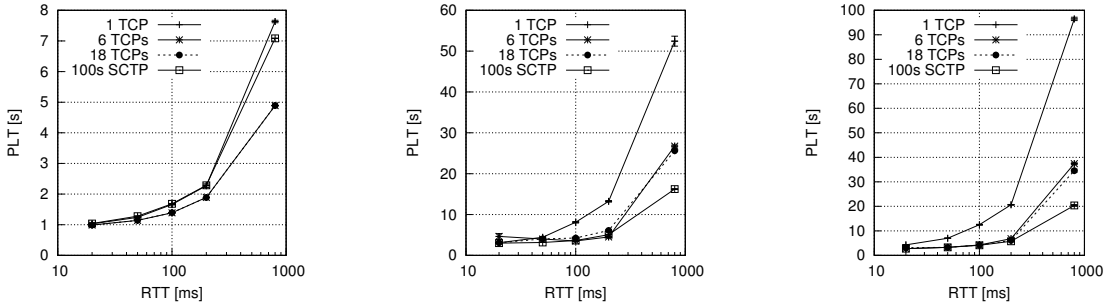
Next, we examine the impact of processing times influences on the PLT, Figure 4.

The additional processing time does not significantly increase the PLT of a single connection (1 TCP). The request overhead for each resource dominates. Parallelism eliminates this, therefore the processing delay has a direct impact on the PLT (Figure 4), for both parallel flows and multi-streaming, since it resulted in greater temporal dependency between resources from the web model [12]. These results may present an upper bound, since we expect advances in web page design and client processing since this model was published.

C. Impact of loss

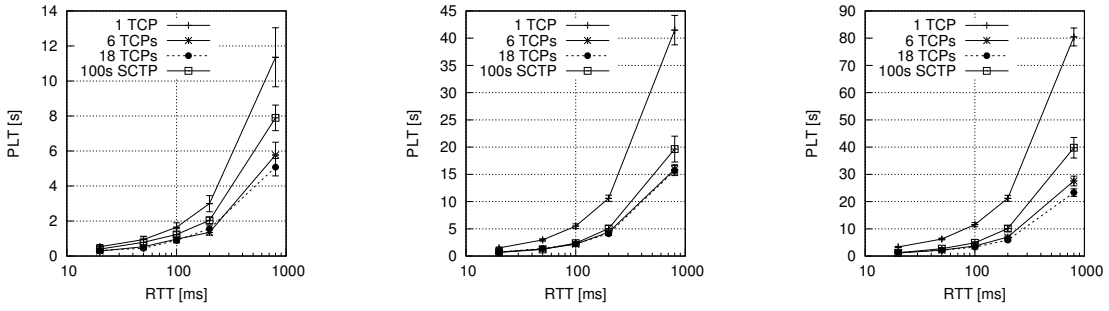
Our results also consider packet loss (e.g., from link effects such as interference on wireless). Figure 5 therefore considers the impact of a simple loss model on the PLT. Loss for a single TCP flow (1 TCP), results in head of line retransmission delay and reduced congestion window. The PLT is reduced by parallelism. Only the TCP connection(s) that experience loss are impacted by loss recovery, the throughput of other parallel flows is unchanged.

In contrast, using multi-streaming head-of-line blocking only impacts a stream the experiences loss, although any loss impacts the congestion window for all streams sharing an association. This more conservative congestion control, results

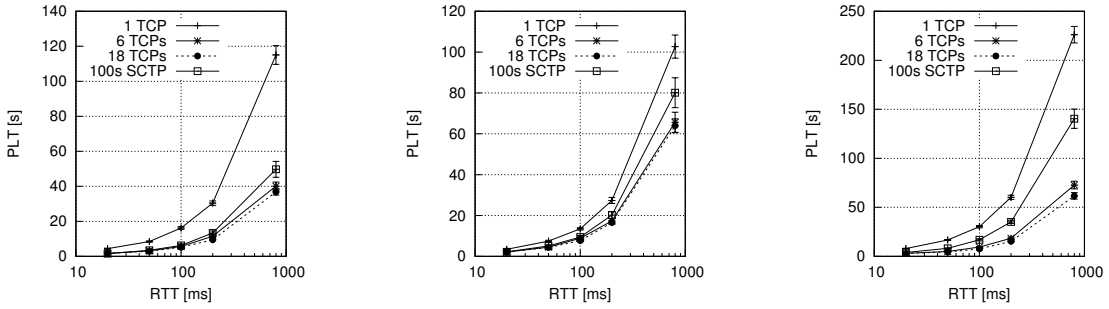


(a) google (b) amazon (c) mediafire

Figure 4: PLT for 10 Mbps capacity, no loss, with processing time.



(a) google (b) dmm (c) siteadvisor.com



(d) amazon (e) pinterest (f) mediafire

Figure 5: PLT for 10 Mbps capacity, 1.5% packet loss, without processing time.

in a higher PLT. (Note: the authors understand that this result could be different if loss was a result of congestion, where reducing the overall capacity consumed by a web client could help reduce future loss).

D. Discussion of the Experiment Setup

A key benefit of multistreaming is the lightweight cost for additional streams, and this allows flows to open as many streams as they need. Our use of SCTP therefore considered a larger number of streams (100) compared to the maximum number of TCP connections (18). The memory allocated by each TCP/SCTP connection consists of a Transmission Control Block (TCB) of about 700 B, which is is more than needed for an SCTP stream (32 bytes) [20]. Although the TCB for SCTP can be twice as large as for TCP, this cost is amortized across the number of streams used.

We did not consider alternative ways to serve the original content, such as domain sharding, or image spriting, which scatters the content across multiple servers. This can have the effect of increasing parallelism, but equally reduces opportunities for multi-streaming. (Using a single origin server is also recognised as best practice for HTTP/2 [21], to exploit the benefits of multi streaming).

Our performance analysis only consider pseudo-random link loss, although we did observe some loss from self-induced congestion. Based on the way parallel connections and a single connection with parallel streams work during packet loss, a different scenario involving concurrent traffic where for example

bottlenecks are not empty when a web request is made would lead to different PLT values. Future research will explore the effect of a range of congestion bottlenecks.

E. From Transport Mechanisms to web Transport Protocol

The approach taken by this paper has been to evaluate transport mechanisms to understand their contribution to web page load time. We used a data-driven workload, because we understood already that the performance would be dependent on the structure of the requested web page. Our results commented on how mechanisms were impacted by the level of parallelism and RTT.

There are many obstacles to introducing new transport mechanisms, and it is even harder to achieve widespread deployment of new transport protocols. The architecture developed in NEAT [22] can provide much needed flexibility to introduce new mechanisms and enables a gradual deployment of a new protocol, not previously possible with existing protocol stack designs. In this paper, SCTP was used as a standard mechanism for multistreaming that may be leveraged by a web client. SCTP could also provide additional benefits if deployed for web clients, including multi-homing reliability in case of path failure.

An important next step for our work would be to understand how transport mechanisms are impacted by more complex workloads. At the time of writing, almost 1/8 of web servers have introduced HTTP/2 support. This makes significant changes to the way the transport is used (building on multistreaming discussed in this paper), and to how resources are mapped to the transport.

V. Conclusion and Future Work

This paper explored key transport mechanisms including multistreaming, parallelism, shared and individual congestion control to evaluate their impact on web performance. The mechanisms were explored across a range of network and application scenarios using a tool developed to replay a set of pre-established web page models. This was used to evaluate the benefit of multistreaming. This was seen to significantly improve overall web performance. Multistreaming enabled rapid utilisation of available link capacity and reduced web load time for web pages with a large size objects or larger web pages, benefiting from shared congestion control. However, multistreaming proved detrimental over a path with high loss. Even a single lost packet in an SCTP connection stalls all of the multiplexed streams over that connection. This also applies to streams in HTTP/2 when a loss happens in underlying TCP. QUIC [23] solves this using UDP as the underlying transport supporting out-of-order delivery – a single lost packet for N concurrent HTTP connections will only stall 1 out of N streams. Besides, losses experienced by the shared congestion of multistreaming limit the growth of the congestion window, and lead to an increase in the page load time.

Future work needs to develop the understanding of the impact of specific transport mechanisms, and the merits and demerits of combining mechanisms within an alternate web transport protocol, for example, comparing SCTP and QUIC. A deeper understanding of the performance implications for HTTP/1.1 can also provide a good technical basis for examining how transport design impacts the performance of HTTP/2.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT). The views expressed are solely those of the authors.

References

[1] Y. Elkahatib, G. Tyson, and M. Welzl, "Can SPDY really make the web faster?" in *2014 IFIP Networking Conference*, Trondheim (Norway), Jun. 2014, pp. 1–9.

[2] M. Butkiewicz, H. V. Madhyastha, and V. Sekar, "Characterizing web page complexity and its impact," *IEEE/ACM Transactions on Networking*, vol. 22, no. 3, pp. 943–956, June 2014.

[3] C. A. Avram, K. Salem, and B. Wong, "Latency amplification: Characterizing the impact of web page content on load times," in *2014 IEEE 33rd International Symposium on Reliable Distributed Systems Workshops*, Oct 2014, pp. 20–25.

[4] X. S. Wang *et al.*, "Demystify page load performance with wprof," in *Proc. of the USENIX conference on Networked Systems Design and Implementation*, 2013.

[5] B. Briscoe *et al.*, "Reducing internet latency: A survey of techniques and their merits," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2149–2196, thirdquarter 2016.

[6] R. Fielding *et al.*, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), Internet Engineering Task Force, Jun. 1999.

[7] R. Secchi, A. Mohideen, and G. Fairhurst, *Evaluating the Performance of Next Generation Web Access via Satellite*. Cham: Springer International Publishing, 2015, pp. 163–176.

[8] ——, "Performance analysis of next generation access via satellite," *Int. J. Satell. Comm. N. (IJSCN)*, vol. 34, no. 6, Dec. 2016.

[9] I. Grigorik, "Making the web faster with HTTP 2.0," *Commun. ACM*, vol. 56, no. 12, pp. 42–49, Dec. 2013.

[10] R. Stewart, "Stream Control Transmission Protocol," RFC 4960 (Proposed Standard), Internet Engineering Task Force, Sep. 2007.

[11] P. Natarajan, P. D. Amer, and R. Stewart, "Multistreamed web transport for developing regions," in *ACM SIGCOMM Workshop on Networked Systems for Developing Regions (NSDR)*, Seattle, Aug. 2008.

[12] X. S. Wang *et al.*, "How Speedy is SPDY?" in *11th USENIX Symposium on Networked Systems Design and Implementation* , Seattle, Apr. 2014, pp. 387–399.

[13] M. Carbone and L. Rizzo, "Dummynet revisited," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 12–20, Apr. 2010.

[14] libcurl — Client-side URL Transfers. [Online]. Available: https://curl.haxx.se/libcurl/c/libcurl.html

[15] phttpget - pipelined http get utility. [Online]. Available: http://www.daemonology.net/phttpget/

[16] phttpget - pipelined http get utility with sctp support. [Online]. Available: https://github.com/NEAT-project/HTTPOverSCTP/tree/multistream

[17] thttpd — Tiny/Turbo/Throttling HTTP Server. [Online]. Available: http://acme.com/software/thttpd/

[18] thttpd with SCTP Support. [Online]. Available: https://github.com/nplab/thttpd/tree/multistream

[19] R. Stewart *et al.*, "Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol," Internet Draft draft-ietf-tsvwg-sctp-ndata, Jul. 2016, Work in Progress.

[20] P. Natarajan *et al.*, "SCTP: An innovative transport layer protocol for the web," in *Proceedings of the 15th international conference on World Wide Web*. ACM, 2006, pp. 615–624.

[21] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540 (Proposed Standard), Internet Engineering Task Force, May 2015.

[22] G. Papastergiou *et al.*, "De-ossifying the internet transport layer: A survey and future perspectives," *IEEE Commun. Surveys Tuts.*, 2016.

[23] Y. Cui *et al.*, "Innovating transport with QUIC: Design approaches and research challenges," *IEEE Internet Computing*, vol. 21, no. 2, pp. 72–76, 2017.