Evaluating the impact of transport mechanisms on web performance for eective web access

Mohideen, A.I.C.; Rajiullah, M.; Secchi, R.; Fairhurst, G.; Brunstrom, A.; Weinrank, F.

Published in:
Journal of Network and Computer Applications

DOI:
10.1016/j.jnca.2019.04.006

Published: 01/07/2019

Document Version
Peer reviewed version

Evaluating the impact of Transport Mechanisms on Web Performance for Effective Web Access

A. I. C. Mohideen[a] , M. Rajiullah[b] , R. Secchi[a] , G. Fairhurst[a] , A. Brunstrom[b], F. Weinrank[c]

[a]*University of Aberdeen, UK*
[b]*Karlstad University, Karlstad, Sweden*
[c]*FHM, Muenster, Germany*

Abstract

This paper explores the design trade-offs required for an Internet transport protocol to effectively support web access. It identifies a set of distinct transport mechanisms and explores their use with a focus on multistreaming. The mechanisms are studied using a practical methodology that utilise the range of transport features provided by TCP and SCTP. The results demonstrate 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 guiding the design of a web transport protocol. Performing this analysis at the level of component transport mechanisms enables the results to be utilised in the design of new transport protocols, such as IETF QUIC.

1. Introduction

This paper explores the transport protocol mechanisms required to realise a modern high-efficient web client. The original specification of HTTP/1.0 serialised the web requests onto a single transport connection, that was assumed to be offered by TCP and originally supported simple web pages with text and a few images. However, web pages have evolved into large highly complex structures [1] comprising a collection of inter-dependent resources. Recent studies [2, 3, 4] have found that the dependency graph for web page resources (and corresponding scheduling order) play a significant role in determining the overall web performance. The order of delivery and processing can therefore be expected to impact the time to display a page, and it is important to understand how transport mechanisms contribute to overall performance. A problem known as Head of Line Blocking (HoLB) occurs when the chain of processing is delayed while waiting for a critical resource to be received over a transport connection [5]. HoLB plagued the performance of early web clients.

To address these problems, various techniques have been employed to accelerate page download [5]. One approach increases the parallelism of resource download, i.e., requesting an HTTP resource while other resources are being downloaded. Therefore, since early specifications of HTTP/1.1, browsers have used a number of TCP connections per server (e.g., the current default is six in Mozilla Firefox and Google

Chrome) and have often adopted a proactive policy for connection management, including closing/reopening slow TCP connections and sometimes requesting the same resource over multiple connections. In addition, servers often choose to distribute webpages across multiple domains (even for the same origin content), a practice known as sharding. A client opens multiple connections for shared content [6]. This further increases the required number of simultaneous transport connections.

Although parallelism has benefits, introducing a large number of transport connections is not without drawbacks. First, the client-server session may experience a large number of connections that do not utilize the full capacity of a path (e.g., a connection may transfer 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 connections reduces the ability to provide congestion control, making web traffic more aggressive towards other competing traffic [7, 8, 9]. Even so, it is still common for HTTP/1.1 clients to use multiple parallel connections to the same web server [10]. One reason for the continued use of parallel connections stems from the stream-oriented design of the TCP transport protocol, which does not provide mechanisms to support sending multiple objects over a single flow.

A number of TCP optimisations have also emerged to improve web performance (larger Initial congestion Window (IW10) [11], TCP Fast Open (TFO) [12], Recent ACKnowledgment (RACK) [13], etc.). However, these optimisations focus principally on the initial congestion control behaviour of TCP, rather than on managing the concurrent transmission of web resources. For example, IW10 propose an increase to the initial TCP congestion window (cwnd) that can reduce the time to start a web transaction, while TFO proposes eliminates one round-trip from the initial TCP protocol handshake. In its simplest form, each transport connection is closed when the requested resource is received. HTTP/1.1 [15] also allowed a client to keep the transport connection open for subsequent requests (known as HTTP persistence), but not finally widely realised until SPDY [16] and HTTP/2 [17] emerged.

The Stream Control Transport Protocol, SCTP [14] provides an alternative to TCP's linear stream by enabling multistreaming. This provides an alternate way to realise parallelism in the transport layer. SCTP was designed to transport signaling information and has not been widely supported for web use. A multistreaming approach can identify sub-streams and relate these to the objects being transported [18]. Persistence is also a feature of an SCTP Association. This enables SCTP to model the transport behaviour with HTTP/1.1.

HTTP/2 introduced a framing layer that helps bidirectional multiplexing of interleaved requests and responses carried over a persistent TCP connection [17]. This layer is key to address HoLB issues associated to various types of interactive web applications (e.g. webRTC). Immediate Data (I-Data) [19], a recent addition to SCTP, refines this approach to allow interleaving also of the transmission units of application messages. This could provide finer control of multistreaming and improve the parallelism.

While this paper analyses interleaving only at a request/response level, it also explores the impact of parallel scheduling within network nodes. Our objective is to evaluate how web resource transmission parallelism is affected by the interaction

between the transport and multi-queue scheduling mechanisms, such as the recently proposed Flow Queuing (e.g. FQ-CoDel [20]).

The contribution of this paper is three-fold: (a) it uses a web traffic workload based on both a dependency graph and the processing time for HTTP objects at a web client to explore the benefits of multistreaming; (b) it provides new data examining the impact of RTT and bottleneck capacity on web performance; and (c) it seeks to understand the contribution of buffering within the network and the impact of Active Queue Management (AQM) on transport parallelism and multistreaming.

While it examines how recent mechanisms available in TCP and SCTP can enhance the performance of web traffic, a single paper cannot cover all recent transport innovations. For a broader picture, we refer the reader to companion papers [21, 22, 23] that discuss other approaches, including the multipath made available in Multipath TCP (MPTCP) and the Concurrent Multipath Transmission (CMT) of SCTP.

The remainder of this paper is organised as follows: Section 2 describes our web model and test methodology. The experimental tool and the experiment are described in section 3, followed by performance analysis in section 4, the impact of AQM on the transport is discussed in section 4. The paper concludes in section 7.

2. Web Model and Dataset

The analysis explores the performance using a range of transport mechanisms. This requires a representative workload. We utilised a publicly available web performance dataset [24], which provides the number and size of HTTP resources (objects) from 170 recorded web pages. This includes graphs representing the dependency between HTTP resources and their processing time at the client, enabling others to repeat our tests if required.

To characterise the web traffic workload, we categorised the web pages according to the total size of all resources within a page. This total was used to divide each page into one of six bins (size-ranks), labeled A to F, organised so that each size-rank held an equal number of web pages, forming statistically significant groups. Table 1 reports the interval of 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 size-rank, 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. Also, it shows a wide spread distribution in the number of resources

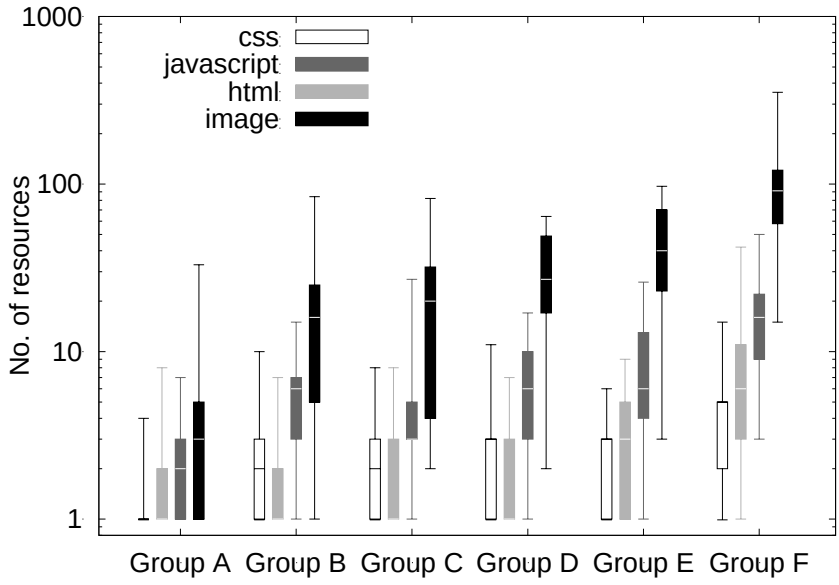


Figure 1 Distribution of number of resources within a web page by MIME type across the six size-ranks.

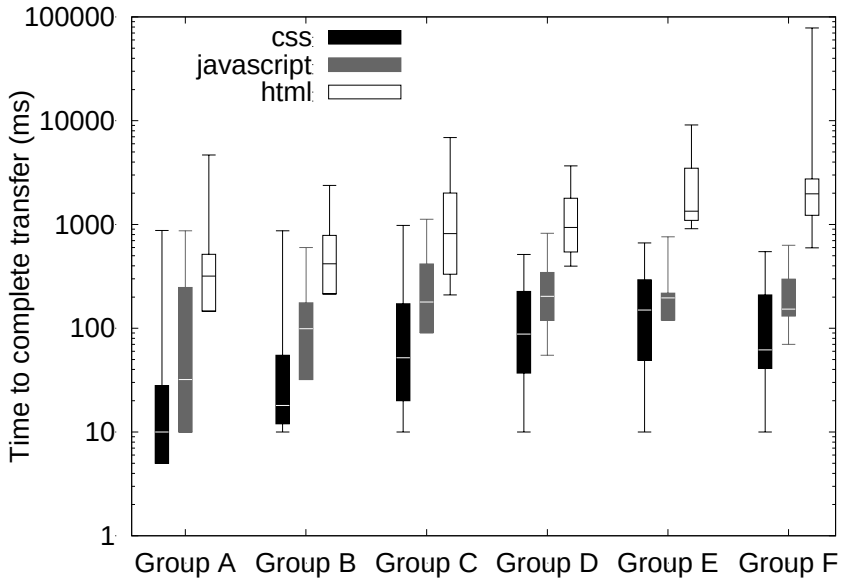


Figure 2 Distribution of time to complete a transfer by MIME type across the six size-ranks.

For example, the number of resources/pages in the smallest size-rank (A) varied between 1 and 39, whereas the largest size-rank (F) ranged between 49 and 228 resources/page. Pages of similar size may have a quite dissimilar composition. Therefore, it may not be sufficient to characterise web pages only by their overall size.

The size of the retrieved resources was also 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 average resource

size has less spread than the distribution of the number of resources, values given within square brackets in Table 1 show the average object size at 95% percentile.

To reduce the number of experimental tests our experiments consider only the webpage with median size for each size-rank. Each size-rank contain all the web pages whose size is between a minimum and maximum. The range of sizes considered are in non-overlapping intervals.

Figure 1 categorises resources by their MIME type, showing the four most common types: text files (HTML), scripts (javascript), style-sheets (CSS) and images (the most common across all size ranks). We 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 contributed less than 2%, including Flash resources, octet-stream and fonts.

Figure 2 shows the distribution of the time spent by the client to complete the transfer of a resource (including computation time). This figure excludes images, because these are terminal nodes in the dependency graph. We observe that for a network path Round Trip Time (RTT) of a few tens of milliseconds, the computation time was often not negligible compared to the time to transmit the object time across the network. In these datasets, the transfer 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 this paper.

3. Tools and Experiment Setup

3.1. Experimental Testbed

Our performance analysis considered two scenarios; 1) a simple path with no competing traffic and predefined patterns of loss, and 2) a path with competing traffic through a network bottleneck. The former reveals the impact of transmission rate and propagation delay, while the latter also considers the impact of a bottleneck and the resulting interaction between transport congestion control and network buffering.

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) [3]
B	119-565	129 (3)	325 (21)	532 (67) [8]
C	566-873	567 (6)	690 (25)	846 (69) [12]
D	874-1242	878 (6)	964 (45)	1183 (82) [14]
E	1243-1945	1286 (24)	1546 (55)	1901(119) [16]
F	1946-3315	2070 (49)	2454 (127)	3309 (228) [15]

Table 1: Webpage size and 5, 50 and 95 percentile of number of resources per size-rank. The page rank of the size is shown in parenthesis. The number of webpage resources of the median case is shown within square brackets.

Our testbed comprised a set of three computers emulating a web client, the 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 netem traffic shaper [25] configured with a bottleneck capacity, delay, buffer size, and packet loss ratio (in scenario 1).

Scenario 2 considered a bottleneck with the default First-In First-Out (FIFO) queuing provided by Linux and the use of Active Queue Management (AQM), controlled via Traffic Control (tc) commands. The AQM testbed used CoDel [26] and FQ-CoDel [20] queue management algorithms and followed the best practices from the bufferbloat community [27]. We followed a methodology described for parameterising the AQM algorithms [28] and choose the buffer size at the bottleneck as 152ms (corresponding to 127 full-sized packets for a 10 Mbps capacity link).

Scenario 2 experiments using Flent [29] to understand the real-time response under load. We created two competing bulk TCP flows that saturate the buffer at the bottleneck for the entirety of each web experiment. The competing flows used Cubic congestion control. This setup was used to measure (at steady state) and study the impact of a congested bottleneck on the web Page Load Time (PLT), as well as to understand the contribution of AQM.

Our analysis included experiments using a range of symmetric paths at 2 Mbps, 10 Mbps and 100 Mbps. Results for 100 Mbps indicated similar relative performance for different transport mechanisms. This was also observed in an empirical study at Google [30]. Results for lower rate paths, at or below 2 Mbps, are known to have a strong dependency on the speed of the bottleneck, the effect of competing traffic on performance, and link scheduling methods, and are not the focus of the present paper. The remainder of the paper therefore focusses on a 10 Mbps bottleneck. Similarly, 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 for both mobile and desktop clients (see Table 3).

The client and server supported TCP (Linux 4.2.0-42 and BSD) and SCTP (BSD). The same IW was used for TCP and SCTP. Each 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 servers, and an experimental IETF specification. The maximum segment size was 1460 B.

The multistreaming web server is described in section 3.3. A custom client emulated a HTTP/1.1 browser (section 3.2), enabling requests with either a number of parallel TCP connections (1, 6 and 18[1]) or a single SCTP association. The number of streams is not a significant factor when using a multistreaming protocol, and we allowed up to 100 parallel SCTP streams. The cost of opening a stream is further discussed in Section 4.4. The key experiment parameters are summarised in Table 2.

Experiment parameters		
Category	Factor	Range/value
Network	RTT Bottleneck Capacity Packet loss ratio	20, 50, 100, 200, 800 ms 10 Mbps No loss, 1.5%, 3%
TCP/SCTP	IW CWND validation No. parallel TCP flows No. SCTP streams	client (IW 3), server (IW 10) no 1, 6, 18 100

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

Table 2: Parameters used in our experiments.

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

Table 3: Path RTT from data used in our experiments

3.2. pReplay Web client

Web requests were generated using the pReplay tool [31], developed in C and based on Epload [24]. Epload generates a resource dependency graph of the web page being analysed from a packet capture. The dependency graph can be used to mimic the behaviour of the web browser by generating a request for a web resource only when trigger events of that have completed. pReplay uses libcurl [32] to replay HTTP traces using HTTP/1.1 over TCP or a modified version of phttpget [33] extended to support SCTP [34].

The tool uses a dependency graph in JSON format that represents the resource requests and computation times required to process javascripts, 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 for the relevant URL. The tool optionally simulates the computational activity by waiting for a time determined by the graph. Once an activity completes, pReplay checks whether all the dependent activities have also completed and only then commences the next activity. It finishes when all activities in a dependency graph have been visited.

3.3. Lightweight Web Server

We used a server modified from the lightweight web server thttpd (tiny HTTP deamon) [35] supporting HTTP/1.1. The server is based on a patch that allowed thttpd to run over SCTP [36], but only enabled web traffic to use a single stream for each SCTP Association. This implementation is based on the FreeBDS library libsctp which provides a Reno-like congestion control and includes support for selective acknowledgments. The server was extended to enable parallel multistreaming [36], to enable algorithms to allow 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]).

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

Table 4: Statistics for the web pages forming the workloads used in the experiments.

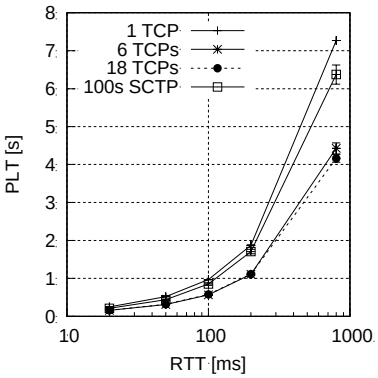
4. Web performance using an Unloaded Bottleneck

This section contains a systematic study of web PLT when 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 using 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.

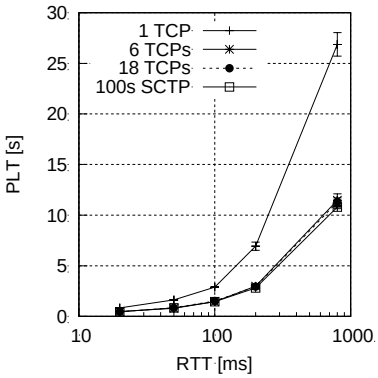
Results are presented for websites at the 50th percentile from our web model (Section. 2) as described in Table 4. The dataset processing time [24], was used as an upper bound for analysing the impact of processing time. Since client platforms continue to evolve in the way that resources are parsed, we expect this result to represent an upper bound on processing time. We therefore also plot the PLT with no additional processing time, to present a minimum bound.

4.1. Impact of Parallelism at the Transport

Figure 3 shows the impact on PLT for the selected number of parallel TCP connections compared to a single SCTP connection. The picture presents the case of 1 TCP flow per web connection (i.e. when parallelism is not allowed), the case of 6 TCP flows per connection as a number represented by most of web browsers, and a large number of TCP flows (18) where that web connection performance is not affected by the number of parallel flows. SCTP has no limitation in the number of parallel streams it can open (100 was chosen to avoid interfering with the ability of SCTP to create new streams). The results in the figure show the simplest with no processing time dependency and no emulated link loss (tail drop loss from FIFO router buffers was observed in some experiments).



(a) google



(b) dmm

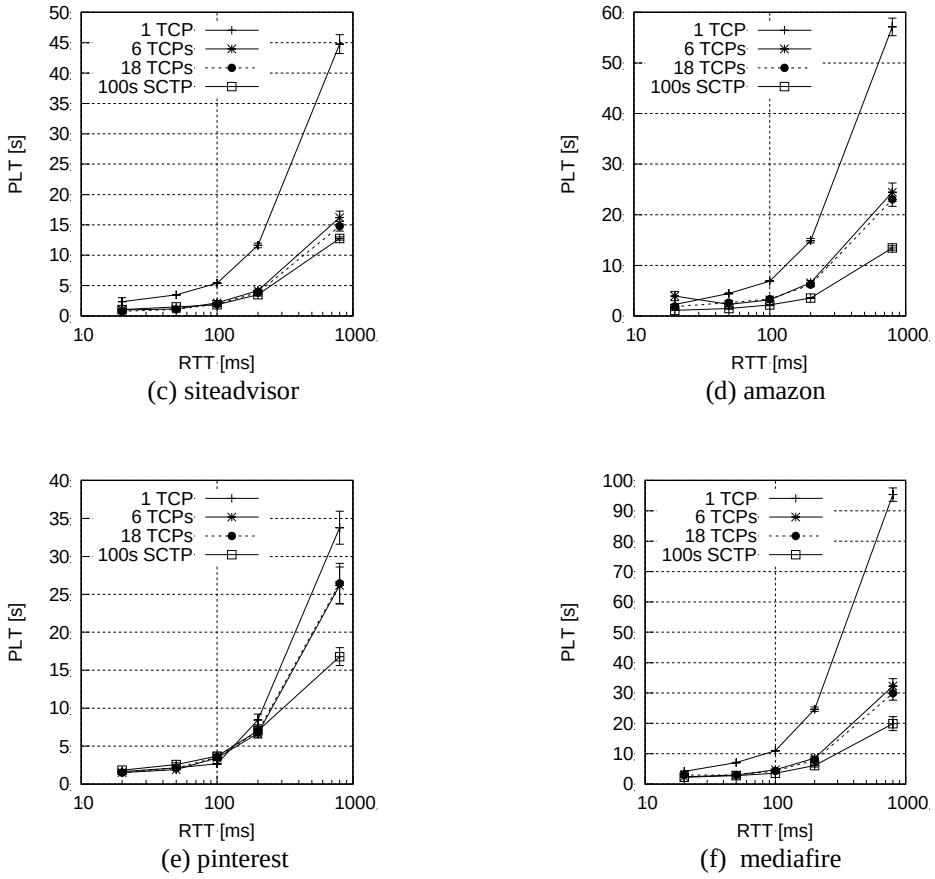


Figure 3: PLT for 10 Mbps capacity, no loss, without processing time.

Each transport pipe independently performed start-up, congestion control and loss recovery. Each TCP transport connection transferred just one single web resource. Our experiments considered two ways in which this parallelism could be introduced: First, using parallel TCP connections (each independently managing congestion control) or second using multiple SCTP streams (where all streams shared a single congestion controller).

Figure 3 shows that when enabled, parallelism used one transport pipe to carry each resource. This reduced the number of consecutive RTTs required to complete transfer of a web page, reducing the PLT. An exception may be seen in Figure 3e, where, one, six and eighteen TCP connections and a multistream SCTP association both have an almost similar PLT (up to the 100ms RTT case). Pages of large size with fewer resources (large average object size); the *pinterest* workload with 6 objects of 258 KB average object size (see Table 4), has a similar PLT with parallelism. We discuss this special scenario later in the AQM section.

The benefits of parallelism may come at a cost because:

- For a transport protocol with an independently managed congestion control (e.g. TCP), a higher sending rate can induce congestion at the bottleneck link leading to collateral damage to other flows that share the bottleneck.

- For a multistreaming transport protocol that uses a shared congestion control (e.g. SCTP), each stream contributes to the capacity used by the association. This increases growth of the cwnd, reducing the PLT (Figure 3). When congestion is experienced, a

multistreaming protocol will reduce its rate, reducing collateral damage to other flows. However, this has a negative impact on the protocol's throughput.

In most cases, (except for the google sites in Figure 3a), a multistreaming approach provided a smaller PLT than when N parallel TCP pipes were used. The latter consume more overhead to set-up parallel connections, and create self-induced congestion from concurrency.

For small pages, (e.g. google in Figure 3a), the combined IW provided by N TCP connections have benefit compared to the single shared IW with multistreaming. However, again at the risk of more collateral damage.

The PLT for all the web workloads shown in Figure 3 is higher for a larger path RTT. However, multistreaming has benefit for a higher RTT, where the connection overhead becomes important (e.g., in Figure 3f, the PLT increases over 282% using 18 TCP parallel connections, compared to 229% using multistreaming for an RTT of 200 ms to 800 ms).

Web page structure also impacts the PLT. When there is no parallelism, the number of resources influences the PLT more than the overall page size. This may be seen in Figure 3d, for 1 TCP, where the Amazon workload (with a larger number of smaller resources) complete much later than the Pinterest workload in Figure 3e (with fewer larger resources, Table 4). Therefore, the number of resources and the average size of the objects have more impact on the overall performance than the total size. Parallelism alleviates this by reducing the delay from HoLB for pages with many resources (e.g. the PLT for Amazon is lower than that for Pinterest when either multistreaming or N parallel TCP connections are used).

4.2. Impact of processing time at the client

This section examines the influence of processing time on the PLT, Figure 4. The additional processing time does not significantly increase the PLT when using a single connection (1 TCP), where the request overhead for each resource dominates. Parallelism eliminates this overhead, therefore the processing delay has greater temporal dependency between resources from the web model [24] and can be observed to have a direct impact on the PLT (Figure 4). This demonstrates the importance of reducing processing delay when designing web clients, although the authors did not have any way to evaluate how the model for processing delay would have changed if a modern web client had been used.

4.3. Impact of loss

Our results also consider the impact of a simple loss model on the PLT (e.g., from link effects such as wireless interference), see Figure 5. Loss for a single TCP flow (1 TCP) results in a HoLB delay and reduces the cwnd (reducing throughput). Parallelism reduces the PLT when using TCP, because a loss only impacts the transport connection and the throughput of other parallel flows is unchanged.

When using multistreaming, loss only results in HoLB for the (sub)stream that experiences a loss. However, any loss also impacts the cwnd shared by all streams in an SCTP Association. The shared congestion control reacts more conservatively, and results in a higher PLT. If the loss was a result of congestion, this result could have been different, since then reducing the overall capacity consumed by a client could also help reduce future loss and ultimately reduce the PLT.

4.4. Discussion of Multistreaming Analysis

A key benefit of multistreaming is the lightweight cost for adding a new stream, which allows a client to open as many streams as they need. Our use of SCTP therefore considered a larger maximum 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 much more than needed for a SCTP stream (32 B) [37]. Although the TCB for an SCTP association can be twice as large as for TCP, this cost is amortized when multiple streams are used.

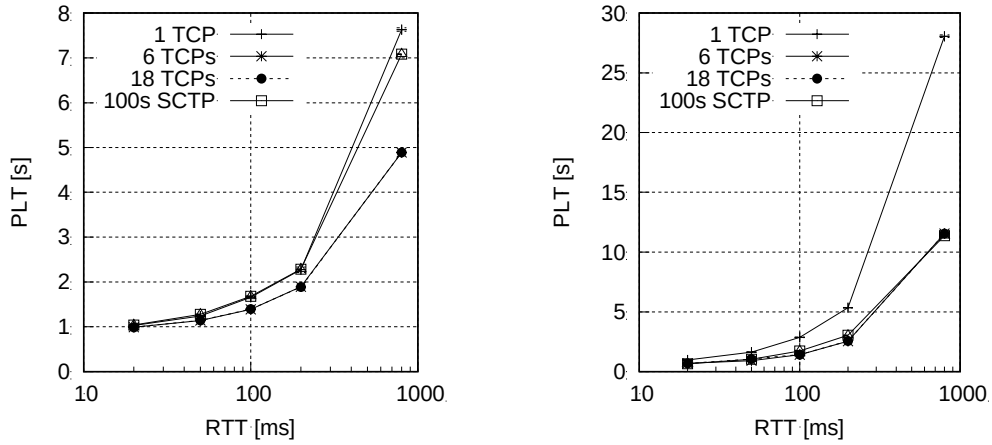
Our performance analysis only considered a scenario with added link loss, although we did observe loss due to from self-induced congestion. We also did not consider alternative ways to serve the original content, such as domain sharding (to scatter the content across multiple servers), or image spriting[2]. These can change the opportunities for parallelism, but reduce opportunities for multistreaming. Using a single origin server has been recognised as best practice for HTTP/2 [17], to exploit the benefits of multistreaming.

Our analysis in Section 4 has shown:

- The number of web resources and the average size of a web resource impact the transport much more significantly than the total page size. The performance of short-lived flows (small objects) is limited by the growth of the cwnd and is a direct function of path delay.

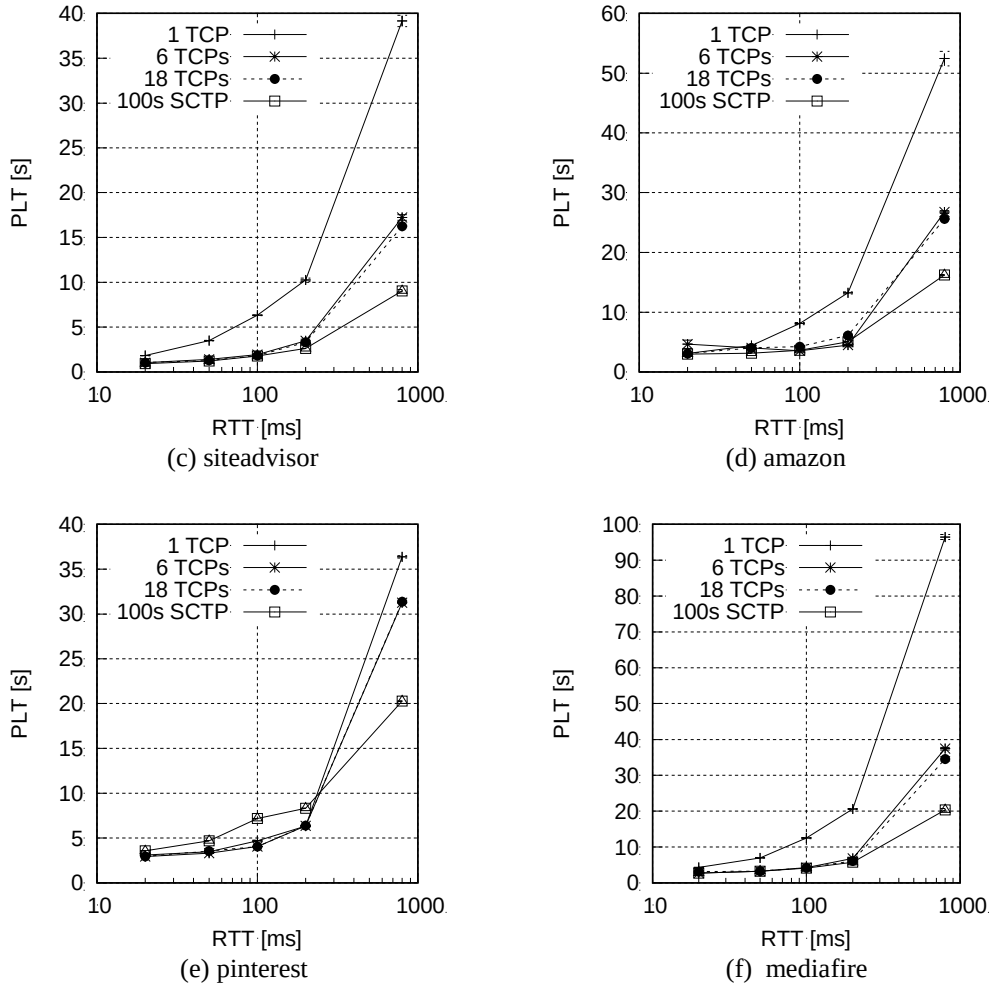
- Paths with a shorter RTT may be expected to experience more rapid loss recovery, e.g., TCP Cubic provides one recovery per pipe (no multistreaming). This is particularly important for small resources. However, there is also a pathology that can result in loss recovery based on the Retransmission Time Out (RTO) [38], which can significantly increase PLT.

- The inter-dependency (and processing time) between web resources reduces web performance when using multistreaming. This behaviour is not limited to TCP, for instance, Control block sharing [39] with shared bottleneck detection [40] could also result in a similar behaviour.



(a) google (b) dmm

[2] Image spriting is a technique increasingly used in web design to send a group of separate images as a single cluster. Using a single transmission unit for all images reduces the number of HTTP requests and accelerates the web transaction.



(c) siteadvisor (d) amazon

(e) pinterest (f) mediafire

Figure 4: PLT for a link with 10 Mbps capacity, no loss, including client processing time.

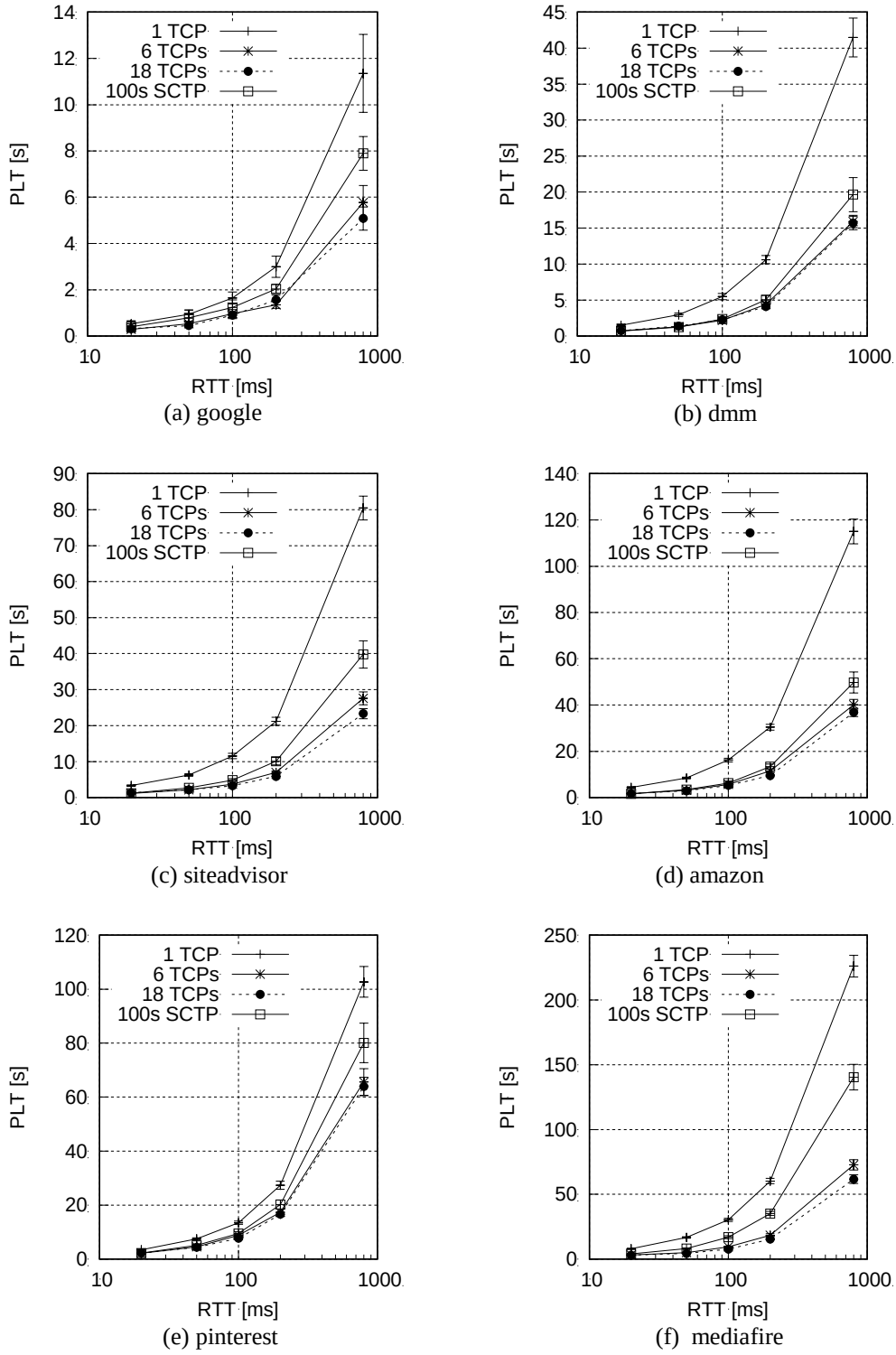


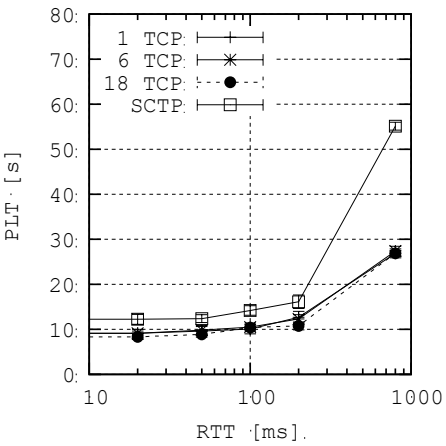
Figure 5: PLT for a 10 Mbps capacity link, 1.5% packet loss, without client processing time.

5. Exploring Shared Congestion Bottlenecks

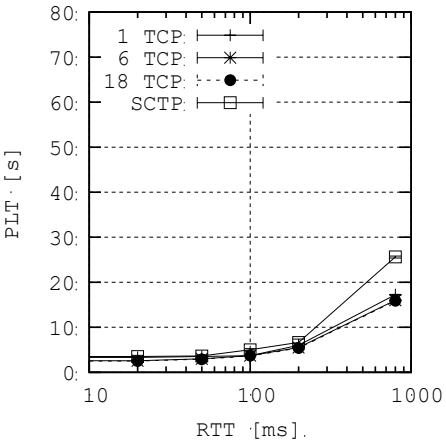
This section evaluates the impact of bottleneck congestion on PLT. We evaluate three instances of bottleneck buffer management: A drop-tail FIFO queue, a Controlled Delay (CoDel) queue [26], and a queue managed by flow-queuing CoDel (FQ-CoDel) [20]. We report results for the *Pinterest* and *Mediafire* workloads, which are two large pages in our dataset with very different composition pattern: there are few large objects in the *Pinterest* workload and many small objects in the *Mediafire* workload. Both CoDel and FQ-CoDel are forms of AQM. The bottleneck was loaded by including two long-running bulk TCP flows with the web page download.

5.1. Drop-Tail FIFO Bottleneck

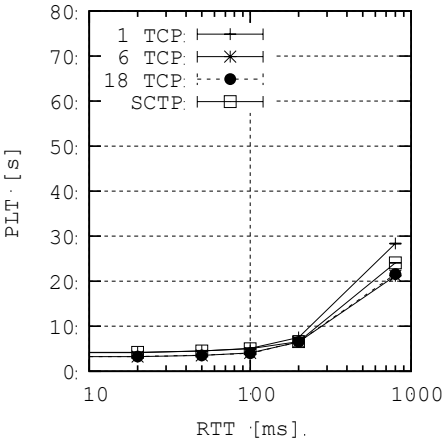
Figure 6a shows the PLT for the *Pinterest* workload. This consisted of 6 resources using a congested shared FIFO buffer. Since the *Pinterest* workload consists of few relatively large objects (around 250 kB), the data sufficient to allow the congestion controller to reach a steady-state. Thus, the PLT is largely dominated by the available capacity and small performance differences are observed in the case of 1, 6 and 18 TCP flows. The small performance loss of SCTP with respect to the multiple TCP case should be attributed to the lack of optimisations in our SCTP implementation, rather than to the inability to use parallel flows. The different interaction between transport and network layers is a small performance gap.



(a) FIFO

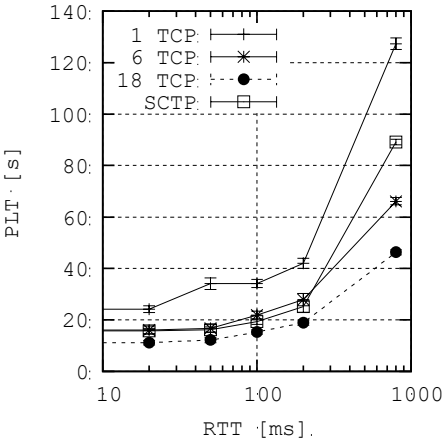


(b) CoDel

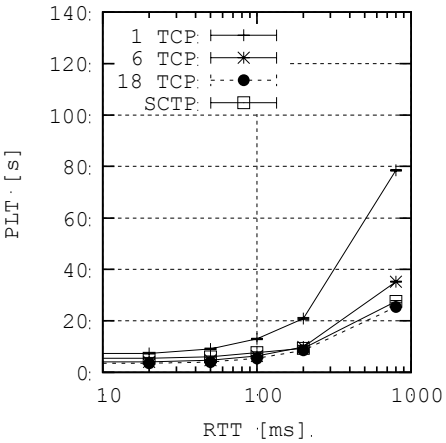


(c) FQ-CoDel

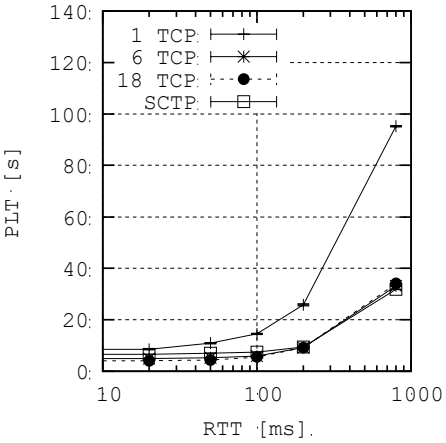
Figure 6: PLT for 10 Mbps capacity, Pinterest workload, with a congested bottleneck.



(a) FIFO



(b) CoDel



(c) FQ-CoDel

Figure 7: PLT for 10 Mbps capacity, Mediafire workload, with a congested bottleneck.

Pinterest PLT (s)				
	1 TCP		6 TCP	
RTT (ms)	IW3	IW10	IW3	IW10
0	8.9	7.7	8.8	8.0
20	10.5	9.1	11.3	9.1
50	10.6	9.6	11.9	9.8
100	15.2	10.0	15.6	10.5
200	16.7	12.7	20.5	12.3
800	50.9	26.9	60.6	27.4
Mediafire PLT (ms)				
RTT (ms)	IW3	IW10	IW3	IW10
0	19.5	20.2	16.7	14.1
20	26.7	24.2	21.2	16.0
50	29.1	34.1	22.6	16.7
100	40.1	34.2	29.7	21.9
200	47.1	42.0	39.2	27.9
800	134.7	127.4	113.0	66.1

Table 5: Comparison of average PLT with IW3 and IW10 for the *Pinterest* and *Mediafire* workloads

5.1.1. Effect of the page fragmentation

The PLT for the *Mediafire* workload in Figure 7a illustrates the effects of page fragmentation on parallelism and multistreaming. This workload has many more objects than the *Pinterest* workload and smaller on average. As already observed in the unloaded bottleneck scenario shown in Figure 4, the PLT for the *Mediafire* workload reduces for a larger RTT. However, a large RTT reduces the multistreaming performance (SCTP) more than for parallel TCP connections, leading to a situation reversed with respect to the one observed without competing traffic (Figure 3f). This effect, which is also visible in Figure 5f, can be attributed to packet drops that occurs shortly after a flow starts. This results in a significant reduction of cwnd. The reduced cwnd continues to have impact for the remainder of the flow duration, increasing the time to download the object and any subsequent object using the same stream. Thus, if the transport consists only of a single congestion controlled stream, the entire transmission is slowed down. Conversely, a server that can choose among several parallel flows, can schedule to deliver resources to best use flows that were not penalised by early packet loss.

This effect is observed in SCTP where the congestion control is not only shared between all concurrent flows, but also persistent across all objects of the same page. A similar effect would be seen if the TCP flows were to be used persistently to sequentially request multiple objects (e.g., as permitted with HTTP/1.1 or HTTP/2).

5.1.2. Effect of the initial congestion window

The size of the IW can have an important effect on performance. Table 5 compares the average PLT (over ten runs) as a function of the RTT for the *Pinterest* and *Mediafire* workloads when a client has an IW of 3 segments (IW3) and an IW of 10 segments (IW10). An IW10 allows up to 15 KB of data to be sent in the first RTT of transfer, reducing the PLT. The results in Table 5 show a gain with IW10 larger than two RTTs. This due to the long

transient period in congested conditions. Since many loss cycles are required to converge to steady-state, starting from a smaller cwnd may have an important impact on PLT.

Important gains were therefore were observed using IW10 with both the *Pinterest* and *Mediafire* workloads. A transport using multistreaming shares one IW among all the (sub)streams and may miss the opportunity of faster startup [8], but releasing more than 10 segments into the network is not recommended to avoid collateral damage [41], unless server-side pacing can be used.

5.2. Web performance using CoDel

The CoDel [26] AQM algorithm limits the queuing delay at the bottleneck link by measuring the queuing time of packets in the network buffer and maintaining a target for the queuing delay that evolves over a pre-set interval. If the queuing delay exceeds the target over the pre-set interval, packets are dropped from the head of the queue until the queuing delay drops below the target. The default values for the target and interval are respectively 5 ms and 100 ms.

The PLT for the *Pinterest* and *Mediafire* workloads is significantly improved for both TCP and SCTP when the bottleneck uses CoDel compared to FIFO (Figure 6b and 7b). The smaller path RTT under load allows faster delivery of data and helps each flow to more quickly grow its cwnd. In the case of the *Mediafire* workload, the old/new flow feature of the Codel scheduler helped performance. The consistent PLT observed for an RTT less than 200 ms in Figure 6b and 7b is a result of this faster control-loop.

CoDel reduces the impact of other flows on the progress of a specific flow. In this way, it can reduce the time to complete a retransmission when multiple TCP flows are being used as demonstrated in Figure 6b and 7b. When a single stream is used, two RTTs are required for the cwnd to grow to send an object on average (33 KB) from the *Mediafire* workload. However, CoDel reduces the queuing delay and hence the RTT, allowing faster growth of cwnd and faster retransmissions compensating for the more aggressive drop policy in CoDel. The advantage of a web transport using a path with AQM is clearly visible in both webpages analysed.

While CoDel effectively improves performance with respect to FIFO, the PLT for the *Mediafire* workload with a single TCP connection remains still high (about 20 s when the RTT is 200 ms). As explained in Section 4, parallelism or multistreaming is needed to improve the performance of webpages with multiple objects.

Transport Mechanisms				
System	Mechanism	TCP	SCTP	QUIC
Transmission	a. TCP Fast-Open TFO (RFC7413)	X	-	X
	b. Multistreaming (RFC4960)	-	X	X
	c. Interleaved Multistreaming (draft)	-	X	X
	d. Per stream flow control (RFC4960)	-	-	X
Loss Detection and Recovery	a. SACK (RFC2018) (RFC5681)	X	X	X
CC Algorithm	a. TCP Cubic	X	-	X
	b. TCP Reno (RFC5681)	X	X	X
Congestion Control	a. IW10 (RFC6928)	X	X	X
	b. New Reno Fast Recovery (RFC3782, RFC6582)	X	X	X
	c. ECN (RFC3819)	X	-	-

. Table 6: Set of Transport Mechanisms used in the Experiments.

5.3. Web performance using FQ-CoDel

FQ-CoDel [20] is a hybrid algorithm that implements CoDel the algorithm on the sub queues of a FQ scheduler. The scheduler uses a five-tuple hashing algorithm to enqueue packets onto sub-queues, and a deficit round robin scheduler to dequeue the packets from sub-queues. The FQ-CoDel mechanism, therefore, promotes flow byte-based fairness of parallel flows sharing a common bottleneck. In this respect, the method mirrors at the network layer the parallelism discussed previously at the transport layer.

The PLT for the *Mediafire* and *Pinterest* workloads are similar when the RTT is below 200 ms using both TCP and multi streaming when there is some form of parallelism. Many of the differences evident when using FIFO or simple CoDel are reduced or eliminated when the bottleneck is controlled by FQ-CoDel. The qualitative behaviour of web flows when using FQ-CoDel is similar to the one with CoDel. (Figure 6c and 7c).

A single TCP flows perform worse using FQ-CoDel than CoDel. A single SCTP Association does not derive benefit from using FQ-CoDel. This could in some cases be due to the lower RTT under load but is likely to be more significantly impacted by the lack of collateral damage from the traffic with which it shares the bottleneck. Our results show that CoDel performs similarly to FQ-CoDel for web. This indicates that the presence of flow queuing may not be essential to boost the PLT performance, a conclusion also found in previous research [28].

6. From Transport Mechanisms to a New Web Transport Protocol

This paper used established open data to produce workload models that have been used to help understand the desirable transport protocol features to support for web traffic. While there is a growing diversity of web content, this approach necessarily restricted the range of web pages that we studied. However, the insight gained helps explain how specific transport mechanisms impact web transfer performance. It also explores the way transport mechanisms interact with router buffers within the network to influence application performance.

This paper has evaluated mechanisms implemented in the TCP and SCTP protocols. The results illustrate the key benefits and drawbacks of multi- streaming. Decisions about how content is structured and retrieved can have significant impact on the performance of specific transport mechanisms (e.g., small objects can improve performance for TCP parallelism, but other content benefits from multistreaming). as transport mechanisms are introduced, we expect that web content will continue to be optimised to match the capabilities of the transport that is used.

The application performance is impacted by the use of transport mechanisms. Using a persistent transport an application can achieve a significantly lower PLT for a succession of HTTP requests. One visible benefit is when a path with appreciable RTT is used to request many resources to complete a page. TCP Fast Open (TFO) also provides benefit by reducing the cost of sub- sequent connection setup to the same server, eliminating one RTT of delay per connection. This can result in similar connection setup cost for persistent and non-persistent use, but has a marginal effect when using persistent use, because there is only a single connection setup.

Our analysis of transport mechanics is applicable to other transports that also need to work across an Internet path. In particular, the results are presented at a time when the IETF is designing mechanisms for a new web transport, IETF QUIC [42]. Although this transport has its origins in work at Google and an experimental deployment of Google's own QUIC

protocol [42], the present standards activity takes a fresh approach to the design of the transport mechanisms. Table 6 compares key transport mechanisms available in QUIC. While this specification has yet to be standardised, it is clear that IETF QUIC will include mechanism to address lessons learned by the community since HTTP/1.1 using TCP. This includes favouring a single multistreamed approach (which shares congestion state, as in SCTP), rather than a single TCP stream or multiple parallel transport sessions (currently the norm). This also matches the persistent reuse of open connections (standardised for TCP in HTTP/2 [17]). Loss detection is expected to be different to either SCTP or TCP, but it is recognised needs to be designed to eliminate head of line blocking, with opportunities to closely integrate with a new web framework based on HTTP/2.

Future work can build upon the baseline analysis presented. When QUIC techniques have been standardised, this work could extend the methodology to compare performance with the results presented here. This analysis could be expanded to consider a wider range of web content. To explore the benefits of new techniques the methodology could also be extended to consider the time to first paint or fold time, to provide metrics that can evaluate new latency reduction strategies. This analysis needs to be performed with care, because the overall benefit will depend on multiple factors. Some techniques have been shown to offer significant benefit, but only when used with particular network scenarios and/or web page constructions. The merits and demerits of combining specific mechanisms also need to be considered when defining a protocol together with how the transport protocol will be used and managed by the networks and managed by the networks over which it needs to operate.

7. Conclusion and Future Work

This paper has provided insight into key transport mechanisms 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 each mechanism and the impact of different styles of web page.

Our results show the effects of multistreaming, parallelism, shared and individual congestion control. We show an appropriate choice of mechanism can significantly improve overall web performance by enabling 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, transport mechanisms can also have drawbacks (e.g., a single multistreamed connection can reduce performance when used over a path that experience high rates of loss).

Since our analysis seeks to understand component transport mechanisms, our deeper understanding of the performance implications for HTTP/1.1, it also can provide a good technical basis for examining how transport design impacts the performance of new transport protocols, such as IETF QUIC.

8. References

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

[2] M. Butkiewicz, H. V. Madhyastha, V. Sekar, Characterizing web page complexity and its impact, IEEE/ACM Transactions on Networking 22(3), 2014 pp. 943–956. doi:10.1109/TNET.2013.2269999.

[3] C. A. Avram, K. Salem, B. Wong, Latency amplification: Characterizing the impact of web page content on load times, IEEE 33rd International Symposium on Reliable Distributed Systems Workshops, 2014, pp. 20–25. doi:10.1109/SRDSW.2014.16.

[4] X. S. Wang, et al., Demystify page load performance with wprof, Proc. of 10th 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, 18(3), 2016, pp. 2149– 2196. doi:10.1109/COMST.2014.2375213.

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

[7] N. Khademi, D. Ros, M. Welzl, The new aqm kids on the block: An experimental evaluation of codel and pie, IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2014, pp. 85–90. doi:10.1109/INFCOMW.2014.6849173.

[8] 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, SIGCOMM Comput. Commun. Rev. 40(3),2010, pp. 26–33. doi: 10.1145/1823844.1823848.

[9] R. Secchi, A. Mohideen, G. Fairhurst, Performance analysis of next generation access via satellite, Int. J. Satell. Comm. N. (IJSCN) 34(6), 2016, pp. 29-43.

[10] I. Grigorik, Making the web faster with HTTP 2.0, Commun. ACM 56(12), 2013, pp. 42–49.

[11] J. Chu, N. Dukkipati, Y. Cheng, M. Mathis, Increasing TCP's Initial Window, RFC 6928 (Experimental), Apr. 2013, URL http://www.ietf.org/rfc/rfc6928.txt

[12] Y. Cheng, J. Chu, S. Radhakrishnan, A. Jain, TCP Fast Open, RFC 7413 (Experimental), Dec. 2014,URL http://www.ietf.org/rfc/rfc7413.txt

[13] Y. Cheng, N. Cardwell, RACK: A Time-based Fast Loss Detection Algorithm for TCP, Internet Draft, draft-cheng-tcpm-rack-00, Work in Progress, July. 2018, URL https://tools.ietf.org/html/draft-cheng-tcpm-rack

[14] R. Stewart, Stream Control Transmission Protocol, RFC 4960 (Proposed Standard), Sep. 2007, URL http://www.ietf.org/rfc/rfc64960.txt

[15] R. Fielding, et al., Hypertext Transfer Protocol – HTTP/1.1, RFC 2616 (Draft Standard), Jun. 1999, URL http://www.ietf.org/rfc/rfc2616.txt

[16] Google, SPDY: An Experimental Protocol For a Faster Web. URL http://www.chromium.org/spdy/spdy-whitepaper

[17] M. Belshe, R. Peon, M. Thomson, Hypertext Transfer Protocol Version 2 (HTTP/2), RFC 7540 (Proposed Standard), May 2015, URL http://www.ietf.org/rfc/rfc7540.txt

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

[19] R. Stewart, et al., Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol, RFC 8260, Nov. 2017, URL http://www.ietf.org/rfc/rfc8260.txt .

[20] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, E. Dumazet, The flowqueue-codel packet scheduler and active queue management algorithm (Experimental), RFC8290,, Jan 2018, URL http://www.ietf.org/rfc/rfc8290.txt

[21] A. Rabitsch, P. Hurtig, A. Brunstrom, A stream-aware multipath QUIC scheduler for heterogeneous paths, Proc. of ACM CoNEXT, 2018.

[22] J. Eklund, K.-J. Grinnemo, A. Brustrom, Using multiple paths in SCTP to reduce latency for signalling traffic, Commun. Commun. 129, 2018, pp. 184– 196.

[23] K.-J. Grinnemo, A. Brustr¨om, One the use of MPTCP to reduce latency for cloud applications, Proc. of SNCNW, 2014.

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

[25] Hemminger.S, Network emulation with netem Linux Conf, Au, 2005.

[26] K. Nichols, V. Jacobson, Controlling queue delay, ACM Queue 10(5), May 2012, URL http://doi.acm.org/10.1145/2208917.2209336

[27] Best practices for benchmarking codel and fq-codel, URL http://goo.gl/FpSW5z

[28] T. Høiland-Jørgensen, P. Hurtig, A. Brunstrom, The good, the bad and the wifi, Comput. Netw. 89, 2015, 90–106. doi:10.1016/j.comnet. 2015.07.014.
URL https://doi.org/10.1016/j.comnet.2015.07.014

[29] Toke.hj, flent.
URL https://flent.org

[30] *M Belshe*, More bandwidth doesn't matter (much), URL URL http://www.belshe.com/2010/05/24/more-bandwidth-doesnt-matter-much/

[31] preplay.
URL https://github.com/mrajiullah/pReplay-a-browser-emulator

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

[33] phttpget - pipelined http get utility.
URL http://www.daemonology.net/phttpget/

[34] phttpget - pipelined http get utility with sctp support, URL https://github.com/NEAT-project/HTTPOverSCTP/tree/ multistream

[35] thttpd — Tiny/Turbo/Throttling HTTP Server. URL http://acme.com/software/thttpd/

[36] thttpd with sctp support.
URL https://github.com/nplab/thttpd/tree/multistream

[37] 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.

[38] D. N. Cheng. Y, Cardwell. N, Rack: a time-based fast loss detection algorithm for tcp, Internet Draft draft-ietf-tcpm-rack, Work in Progress, July 2017,
URL https://tools.ietf.org/html/draft-ietf-tcpm-rack

[39] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert R. Scheffenegger, CUBIC for Fast Long-Distance Networks RFC 8312, Feb 2018, URL http://www.ietf.org/rfc/rfc8312.txt

[40] S. Ferlin, Alay, T. Dreibholz, D. A. Hayes, M. Welzl, Revisiting congestion control for multipath tcp with shared bottleneck detection, IEEE INFOCOM - The 35th Annual IEEE International Conference on Computer Communications, 2016, pp. 1–9. doi:10.1109/INFOCOM.2016.7524599.

[41] N. Dukkipati, et al., An argument for increasing TCP's initial congestion window, ACM SIGCOMM Comput. Commun. Rev. 40(3), 2010, pp. 27–33.

[42] J. Roskind, QUIC: Multiplexed stream transport over UDP, Google working design document, 2013, URL https://docs.google.com/document/d/1jdKEQMlM7ThDMDalFYFR_9-Yw91PhoBmkAPQcCicX3s/pub

Author Biographies

Dr. A. C. Mohideen is a researcher at the University of Aberdeen and formerly a faculty member of the University of Westminster. He graduated and received his Master's degree from London Metropolitan University in 2007, and his PhD in 2011, from Nagaoka University of Technology, Japan.

Dr. Mohammad Rajiullah is a post-doctoral research fellow the Karlstad University. He received his Licentiate in Computer Science in 2012 from Karlstad University and Ph.D. in November 2014. His research focuses on transport layer issues for latency sensitive applications.

Dr. Raffaello Secchi is a Lecturer at the School of Engineering. He holds a PhD in Traffic Modelling and Control in High Speed Networks from the University of Pisa. His research interests include the analysis and modelling of the TCP transport protocol and resource management of the DVB-RCS Satellite Broadband system. He contributed to national and European Projects, working on simulation, performance evaluation and transport protocol implementation.

Prof. Gorry Fairhurst is a Professor in Internet Engineering at the University of Aberdeen, UK. His research interests include transport protocol design, low-latency Internet communication, and Internet performance measurement. He is an active participant in engineering Internet standards at the IETF and has developed standards in the Internet and Transport areas. He currently chairs the IETF Transport and Services Working Group. He has coauthored over 150 journal and conference papers.

Prof. Anna Brunstrom received a Ph.D. in computer science from the College of William & Mary, Virginia, in 1996. She is a full professor of computer science at Karlstad University, Sweden. Her research interests include transport protocol design, low-latency Internet communication, and performance evaluation of mobile broadband systems. She is currently KaU PI within two H2020 projects and a co- chair of the rmcat IETF working group. She has coauthored over 150 journal and conference papers.

Felix Weinrank received his B.Sc. and M.Sc. in computer science from the Münster University of Applied Sciences in 2012 and 2014, respectively. He is currently a Ph.D. student in the Department of Electrical Engineering and Computer Science of the Münster University of Applied Sciences. His research interests include the SCTP transport protocol, low-latency Internet communication, and network emulation.