Multistreamed Web Transport for Developing Regions

Preethi Natarajan\(^1\), Paul. D. Amer\(^1\) and Randall Stewart\(^2\)

\(^1\)Protocol Engineering Lab, CIS Dept
University of Delaware
\{nataraja, amer\}@cis.udel.edu

\(^2\)Internet Technologies Division
Cisco Systems
rrs@cisco.com

ABSTRACT

A multistreamed web transport has the potential to reduce head-of-line (HOL) blocking, and improve response times in high latency Internet browsing environments, typical of developing regions. In our position paper \[13\], we proposed a design for HTTP over the multistreamed Stream Control Transmission Protocol (SCTP), and implemented the design for non-pipelined (HTTP 1.0) transactions in the Apache web server and Firefox web browser. We have since adapted Apache and Firefox to handle HTTP 1.1 persistent, pipelined transfers over SCTP streams. Initial emulation results over high latency paths reveal that HTTP over SCTP streams benefits from faster page downloads, and achieves visually perceivable improvements to pipelined objects’ response times. Movies comparing page downloads of HTTP/TCP vs. HTTP/SCTP streams can be found on the author’s website \[12\]. The promising results have motivated us to propose a low cost, easily realizable, gateway-based HTTP over SCTP deployment solution to enhance users’ browsing experience in developing regions.

Categories and Subject Descriptors

C.2.5 [Computer-Communication Networks]: Local and Wide-Area Networks – Internet; C.2.6 [Computer-Communication Networks]: Internetworking – Standards; C.4 [Performance of Systems]: Design Studies; Fault Tolerance; Reliability, availability and serviceability.

General Terms: Performance, Design, Human Factors.

Keywords: Developing regions, web response time, head-of-line blocking, transport layer multistreaming, SCTP.

1. INTRODUCTION

HTTP [RFC2616] requires a reliable transport protocol for end-to-end communication. While historically TCP has been used for this purpose, HTTP does not require TCP. TCP offers a single sequential bytestream to a web server. In the case of HTTP 1.1 with persistence and pipelining, the independent HTTP responses are serialized and sent sequentially over a single TCP bytestream. In addition, TCP provides in-order delivery within the bytestream — if a transport protocol data unit (TPDU) containing HTTP response \(i\) is lost in the network, successive TPDUs containing response \(i+n\) (\(n \geq 1\)) will not be delivered to the web client until the lost TPDU is retransmitted and received. This problem, known as head-of-line (HOL) blocking, occurs because TCP cannot logically separate independent HTTP responses in its transport and delivery mechanisms.

Transport layer multistreaming is the ability of a transport protocol to support streams, where each stream is a logical data flow with its own sequencing space. Within each stream, the transport receiver delivers data in-sequence to the application, without regard to the order of data arriving on other streams. The Stream Control Transmission Protocol (SCTP) [RFC4960] is a standardized reliable transport protocol which provides multistreaming. Independent HTTP responses transmitted over different SCTP streams of the same association (SCTP's term for transport layer connection) can be delivered to the web browser without HOL blocking, enabling the browser to download and render multiple HTTP responses in parallel, a.k.a. concurrent rendering. Concurrent rendering presents the user with more content of embedded web objects compared to traditional sequential rendering in HTTP/TCP, especially in environments where HTTP/TCP suffers from exacerbated HOL blocking.

![Figure 1: Internet Connectivity via VSAT Link](image)

While many web users in developed nations experience excellent browsing conditions with minimal end-to-end delays and negligible packet loss rates, a large and growing portion of WWW users in developing nations experience much higher delays and loss rates. Such high delays transpire from low bandwidth and/or high propagation delay links, such as VSAT/3G/GPRS. For a multitude of factors, VSAT links (Figure 1) are the most cost-effective and efficient method of providing Internet connectivity for commercial customers, governments and consumers in developing nations [16]. In addition to propagation delays, sub-optimal traffic routing increases latency of Internet traffic in developing nations [2, 5]. For
example, sub-optimal routing for intra-African traffic results in Internet traffic traversing multiple VSAT links, and/or being routed through North America or Europe, leading to high RTTs [15]. Furthermore, Internet traffic to/from developing regions traverses through lossy paths, and experiences significant end-to-end loss rates [5, 15].

In this work, we focus on concurrent rendering to enable visually perceivable response time improvements in high latency and lossy browsing conditions found in developing nations. Online U.S. shoppers consider 4 seconds as the maximum acceptable page download time before potentially abandoning a retail site [1]. Response times above 4 seconds interrupt the user experience, causing the user to leave the site or system. While web users over high latency and lossy paths in developing nations must be more tolerant to response times, these users will prefer to use a system that provides better browsing experience.

This paper is organized as follows. Section 2 gives an overview of the factors affecting HOL blocking. Section 3 discusses our emulation setup, results and observations. Section 4 proposes a realistic, low cost, gradual deployment solution that enables web users in developing regions to benefit from concurrent rendering. Section 5 discusses related work, and finally Section 6 summarizes the work.

2. Factors Affecting HOL Blocking

We consider the following model to understand HOL blocking in an HTTP 1.1 persistent, pipelined transfer containing N embedded objects (Figure 2):

\[ \text{obj}_i = \text{object }i, 0 \leq i \leq N, \text{obj}_0 \text{denotes index.html, obj}_{1..N} \text{denote the embedded objects in index.html.} \]

\[ \text{req}_i = \text{time when the web client generates the HTTP GET request for obj}_i, \text{and writes the request to the transport layer.} \]

\[ \text{obj}^k_i = \text{ } k^{th} \text{ piece of obj}_i, 0 \leq k \leq M; \text{obj}_i^0 \text{denotes the response header, and obj}_{1..M}^i \text{denote the different pieces of obj}_i. \text{ Note that } M \text{ depends on the size of obj}_i. \text{ In our emulations, we assume all objects are the same size (M).} \]

\[ \text{rsp}_{j}^k = \text{time when transport delivers obj}^k_j \text{ to the web client.} \]

\[ \text{proc}_{i}^k = \text{(req}^k_i - \text{rsp}_{j}^k) \text{ denotes the web client’s processing time for obj}^k_i, \text{ such as decoding and rendering a JPEG image.} \]

In TCP’s sequential rendering, if obj}^k_j \text{ is lost and recovered after } x \text{ time units, pieces of obj}_j (j > i) \text{ could be HOL blocked for } x \text{ time units. Assuming the web client is currently rendering obj}^{k+1}_j, \text{if } (x < \text{proc}_{i}^{k+1}), \text{this instance of HOL blocking does not affect response time for obj}_{i+1}. \text{ Otherwise, the HOL blocking increases obj}_{i+1}'s response time by } (x - \text{proc}_{i}^{k+1}) \text{ time units. Thus, the duration of HOL blocking depends on the loss recovery period, } x. \]

In both TCP and SCTP, the duration of loss recovery based on retransmission after 3 duplicate acks (fast retransmit) takes \(~1\) round-trip time (RTT), and retransmission after timeout expiration (timeout retransmit) takes between the initial retransmission timeout value (RTO) of 3 seconds and the maximum of (1RTT, min RTO (1 second)) [RFC2988]. Note that the loss recovery period increases as the path’s RTT increases. Also, the frequency of HOL blocking increases as the loss rate on the end-to-end path increases. Therefore, HOL blocking could be exacerbated in a high RTT, lossy path.

Apart from end-to-end path characteristics, individual object sizes also influence the degree of HOL blocking. As object size increases, the probability that a piece of the object is lost also increases. Hence, a larger object in a pipelined transfer is more likely to block delivery of subsequent objects than a smaller object would.

3. EVALUATION

Since its inception at the IETF SIGTRAN working group, SCTP has evolved into a general purpose IETF transport protocol with more than 25 implementations, fine-tuned by many interoperability workshops. Reference [13] discusses the HTTP/SCTP streams design and the initial implementation efforts in Apache server and Firefox browser on the FreeBSD SCTP reference implementation.

Adapting a TCP application to work over SCTP streams turned out to be a challenging task. At the time of writing [13], we had implemented HTTP/SCTP for non-pipelined (HTTP 1.0) transactions and had not done any performance evaluations. For this paper, we modified Apache and Firefox to handle persistent and pipelined HTTP 1.1 transactions over SCTP streams, and use the implementations to empirically analyze the benefits of HTTP over multistreamed transport.

The following high latency browsing environments are considered for evaluation [5, 15]. Results for other high latency environments such as High Speed Download Packet Access (HSDPA) links are available in [14].

- 1Mbps link with 350ms RTT (1Mbps.350ms): User in South Asia, accessing a web server in North America over land line.
- 1Mbps link with 850ms RTT (1Mbps.850ms): User in Africa, sharing a VSAT link to access a web server in North America.
- 1Mbps link with 1100ms RTT (1Mbps.1100ms): User in Africa, sharing a VSAT link to access a web server within Africa. The web traffic traverses at least 2 VSAT links.
3.1 Setup

The experiment setup, shown in Figure 3 uses three nodes running FreeBSD 6.1: (i) a client running Firefox browser, (ii) a server running Apache, and (iii) a node running Dummynet [17] connects the server and client. Dummynet’s traffic shaper configures a full-duplex link, with a large queue size between client and server. Both forward and reverse paths experience Bernoulli losses. The loss rates vary from 0%-10%, typical of the end-to-end loss rates observed in developing regions [5, 15].

![Emulation Setup](image)

**Figure 3: Emulation Setup**

RFC2616 recommends web browsers to open a maximum of two transport connections to the same server/proxy. We found disparities between this recommendation and current practice. In Firefox, the number of transport connections to the same server is a tunable parameter, imposed for each tab. Several tabs downloading pages from the same server have multiple (>2) transport connections open to that server, and the same could be true with other browsers. We note that existing research emphasizing the negative consequences of an application opening multiple TCP connections to the same server [RFC3124] applies to an application using multiple SCTP associations as well.

Web transfers over more than one transport connection reduce the number of “in flight” TPDUs per connection, lowering the number of dupacks generated by receiver after TPDU losses [3]. Insufficient dupacks increase the chances of timeout-based loss recoveries, especially in low bandwidth and high latency paths. Recall that timeout recoveries significantly increase the duration of HOL blocking and response times. TPDUs losses lower the cwnd and further decrease the number of “in flight” TPDUs, contributing to more timeout recoveries. Therefore, improving the loss recovery by increasing the number of “in flight” TPDUs per transport connection, i.e., minimizing the number of open connections to a server, could be crucial for improving web response times in lossy, low bandwidth and high latency environments.

In this work, we consider the most simple pipelined transfer scenario. We compare an HTTP 1.1 persistent, pipelined transfer over a single TCP connection with an identical transfer over a single multistreamed SCTP association. Web workload characterization studies reveal that the file size distribution on web servers and the transferred file size distribution are heavy-tailed (Pareto) [18]. Every pipelined transfer in the emulations contains equal sized objects of following sizes: 3KB, 5KB, 10KB, and 15KB. The number of objects in the pipelined transfers (N) also varies: 5, 10, and 15. We believe these values reflect current trends in web pages. For example, the number of embedded images in web pages of online services such as maps.google.com and flickr.com vary from 8 to 20. This number could be higher when clients browse via a proxy. At both client and server nodes, we assume that the transport layer send and receive buffers are not the bottlenecks; they are large enough to hold all data of pipelined transfer.

3.2 Page Rendering Time

In this paper, a web page is considered completely downloaded when Firefox receives the last piece of pipelined transfer from the transport layer (Figure 2). The web page is completely rendered when Firefox processes and draws this last piece on the user screen. In HTTP/TCP sequential rendering, the last piece of data always belongs to the last pipelined object, whereas in HTTP/SCTP concurrent rendering, the last piece of data could belong to any pipelined object. In both schemes, rendering the last piece of an object depends on the throughput of the underlying transport connection.

Page rendering time is defined as the time from when the browser sends the first GET request (index.html), to the time when the last piece of the web page is painted on the screen. Using terminology defined in Section 2,

\[
\text{Page rendering time (T)} = (\text{req}_{M}^{n} - \text{req}_{0})
\]

Our initial hypotheses about SCTP and TCP’s page rendering times were as follows:

(i) Both SCTP and TCP have similar values for their initial cwnd [RFC2414, RFC4960], and employ delayed acks with a 200ms timer. Therefore, we expected both TCP and SCTP’s page rendering times to be identical when no losses occur.

(ii) Both SCTP and TCP employ selective acks (SACKs). Unlike TCP whose SACK info is limited by the space available for TCP options, the size of SCTP’s SACK chunk is larger, limited by the path MTU, and therefore at times contains more information about lost TPDUs than TCP. Also, FreeBSD’s SCTP stack implements the Multiple Fast Retransmit algorithm (MFR), which reduces the number of timeout recoveries at the sender [4]. Therefore, as loss rates increase, we expected the enhanced loss recovery features to help SCTP outperform TCP.

Figures 4-6 show the page rendering times for N=10, averaged over 50 runs with 95% confidence. Similar results for N=5 and 15 can be found in [14]. Interestingly, in all 3 graphs, the results for the no loss case contradict (i), and TCP’s rendering times are slightly (but not perceptibly) better than SCTP’s. Detailed investigation revealed the following difference between the FreeBSD 6.1 SCTP and TCP implementations. SCTP implements Appropriate Byte Counting (ABC) [RFC4960, RFC3465] with L=1. During slow start, the sender increments cwnd by 1MSS bytes for each delayed ack. The TCP stack does packet counting which results in a similar cwnd increase (1MSS per ack) during delayed acks. However, a TCP receiver sends extra acks in the form of window updates, which causes the TCP sender to grow its cwnd more aggressively than SCTP. We expect SCTP to perform similar to TCP when the TCP stack implements ABC with L=1.

As the loss rate increases, SCTP’s enhanced loss recovery offsets the difference in SCTP vs. TCP cwnd evolution. SCTP begins to perform better; the difference even more pronounced for transfers containing larger objects (10K and 15K). For the 1Mbps.100ms case, the difference between SCTP and TCP page rendering times for 10K and 15K transfers is ~6 seconds at 3% loss, and as high as ~15 seconds at 10% loss. For the same types of transfers, the difference is ~8-10 seconds for 10% loss in 1Mbps.350ms scenario.
To summarize, SCTP’s page rendering times are comparable to TCP’s during no loss, and SCTP’s enhanced loss recovery enables faster page rendering times during lossy conditions. More importantly, the absolute page rendering time difference increases, and is more visually perceivable as the end-to-end delay, loss rate, and pipelined transfer size increase.

### 3.3 Response Times for Pipelined Objects

An SCTP association with one stream provides the same concurrency as a single TCP connection, and results in sequential rendering. A multistreamed association provides maximum concurrency for a pipelined transfer when the number of streams equals the number of pipelined objects. Note that concurrent rendering remains unaffected by further increase in concurrency. We use the following metric to capture the concurrency and progression in the appearance of pipelined objects on a user’s screen. Recall terminology from Section 2.

\[ \text{req} = \text{time when browser sends HTTP GET request for index.html.} \]

\[ (\text{rend}_i - \text{req}) = \text{time elapsed from the beginning of the page download (req) to the earliest time when at least P\% of object } i \text{ is rendered.} \]

In sequential rendering, a piece of object \( i \) is rendered only after objects 1 through \( i-1 \) are completely rendered. However, in
concurrent rendering, pipelined objects are displayed independent of each other. $\mu_{\text{Page}}$ is defined as the time elapsed from the beginning of page download to the earliest time when at least $P\%$ of every pipelined object is rendered on the screen, i.e.,

$$
\mu_{\text{Page}} = \max \{\mu_{\text{ren}} - \mu_{\text{req}}; 1 \leq i \leq N\}
$$

Figures 7-9 show the $25\%_{\text{Page}}, 50\%_{\text{Page}}, 75\%_{\text{Page}}$ and $100\%_{\text{Page}}$ values for $N=10$, averaged over 50 runs with 95% confidence. Results for $N=5$ and 15 can be found in [14]. As expected, $100\%_{\text{Page}}$ values for both concurrent (solid points connected by dotted lines) and sequential (hollow points connected by dashed lines) rendering equal the corresponding transport’s page rendering times ($T_i$). Also, the $\mu_{\text{Page}}$ times in concurrent rendering are spread out vs. clustered together in sequential rendering. Concurrent rendering’s dispersion in $\mu_{\text{Page}}$ values signifies the parallelism in the appearance of all 10 pipelined objects.

Both sequential and concurrent rendering schemes’ values are comparable at 0% loss. As loss rate increases, the difference in two rendering schemes’ $\mu_{\text{Page}}$ values increase. In addition, we find that concurrent rendering displays 25%-50% of all pipelined objects much sooner (relative difference ~4 – 2 times for 15K, 10K and 5K objects) than sequential rendering. This result holds true for $N=5$ and 15 as well. In the following subsection, we demonstrate how this result can be leveraged to significantly improve response times for objects such as progressive images, whose initial 25%-50% contain sufficient information for the human eye to perceive the object contents.

### 3.3.1 Concurrent Rendering and Progressive Images

Progressive images (e.g., JPEG, PNG) are coded such that the initial TPDUs approximate the entire image, and successive TPDUs gradually improve the image’s quality/resolution. Via simple experiments, we demonstrate how concurrent rendering considerably improves user perception of progressive images.

---

**Figure 10a:** Concurrent Rendering of Progressive Images (56Kbps, 1080ms; 4.3% loss; $t=7s$)

**Figure 10b:** Concurrent Rendering of Progressive Images (56Kbps, 1080ms; 4.3% loss; $t=12s$)

---

The example web page consists of an initial 1K image of our lab’s logo, followed by 10 progressive JPEG images of world leaders, each of size 10K. Both Firefox/TCP (sequential) and Firefox/SCTP (concurrent) download the example web page over the 56Kbps, 1080ms setup. The full page downloads were captured as movies, and are available online at [12].

In the snapshots shown in Figure 10, both sequential (left) and concurrent (right) runs experienced ~4.3% loss. Both rendering schemes start the download at $t=0s$. At $t=6s$ (not shown), the sequential scheme rendered a complete image followed by a good quality 2nd image, and the concurrent scheme displayed a complete image on the browser window. At $t=7s$ (Figure 10a), sequential rendering displays 2 complete images, vs. concurrent rendering’s 7 partial images, at least 4 of which are of good quality. At $t=12s$ (Figure 10b), sequential rendering displays 4 complete images, whereas concurrent rendering presents the user with all 10 images of good quality. With concurrent rendering, the complete page is rendered only ~$t=23s$. From $t=12s$ to $23s$, all 10 images get refined, but the value added by the refinement is negligible to the human eye. Therefore, the user “perceives” all images to be complete by $t=12s$, while the page rendering time is actually $t=23s$. In the sequential run, all 10 images appear on the screen at $t=26s$.

### 4. IMPACT IN DEVELOPING REGIONS

While concurrent rendering’s initial results promise better response times for web users in developing regions, it is impractical to expect all web servers to provide web over multistreamed SCTP in the immediate future, without which the web users cannot leverage concurrent rendering’s benefits. To address this issue, we propose a realistic, low cost, gateway-based solution that translates HTTP/TCP to HTTP/SCTP streams for easier and localized deployment. The solution assumes that the web browser is capable of HTTP/SCTP, similar to the SCTP-enabled, freely available Firefox browser used in our experiments. The gateway is physically positioned between the server and client, such that, the gateway talks SCTP to clients over a high latency network, and talks TCP to web servers in the outside world. For the architecture shown in Figure 1, the gateway is positioned between the VSAT ground station (on the left) and the Internet cloud. We believe that the “proxy” configuration in the SCTP-enabled Apache server is a good starting point to achieve the gateway functionality at minimal monetary cost [apache.org].

At a minimum, the gateway solution should provide faster page rendering than HTTP/TCP. This solution can be extended to further enhance pipelined objects’ response times. For example, the gateway could use batch image conversion software [9] to convert embedded JPEG/PNG images to the corresponding progressive versions before forwarding them to the clients. Image conversion at the gateway takes on the order of milliseconds per image, but can improve a user’s response times on the order of seconds.

### 5. RELATED WORK

Significant interest exists for designing new transport and session protocols that better suit the needs of HTTP-based client-server applications than TCP. Several experts agree (for instance, see [8]) that the best transport scheme for HTTP would be one that supports datagrams, provides TCP compatible congestion control on the entire datagram flow, and facilitates concurrency in GET requests. WebMUX [7] was one such session management protocol that was a product of the (now historic) HTTP-NG working group [10]. WebMUX proposed using a reliable transport protocol to provide
web transfers with streams for transmitting independent objects. However, the WebMUX effort did not mature.

Reference [6] proposes the use of Structured Stream Transport (SST) for web transfers. SST (proposed after [13]) functions similar to SCTP streams by extending TCP to provide multiple streams over a TCP-friendly transport connection. Simulation-based evaluations in [6] show that SST provides similar page download times as TCP. The primary contribution of a multistreamed web transport is the reduction in HOL blocking, which is the focus of our work. Using real implementations, we show that reduced HOL blocking in HTTP over multistreamed SCTP results in visually perceivable improvements to individual objects’ response times in browsing conditions typical of developing regions. Also, we note that SCTP is a standardized IETF protocol with many fine-tuned kernel space implementations.

Apart from new session and transport protocols other sender-side techniques to reduce HOL blocking among web objects include Congestion Manager [19] and TCP Session [20]. However, these techniques require the client to open multiple transport connections, potentially increasing resource requirements at a web server than a multistreamed transport connection (discussed in detail in [13]).

Content Delivery Networks (CDNs) replicate web content across geographically distributed servers, and reduce response times for web users by redirecting requests to a server closer to the client [11]. Unfortunately, little research exists on the prevalence of CDNs for content providers and web users outside of developed nations. Also, CDNs cannot lessen web response times when latency is due to (i) propagation delay and/or low bandwidth in the last hop, as is the case in developing regions, or (ii) sub-optimal traffic routing that increases end-to-end path RTTs.

6. SUMMARY
We examined the effects of HTTP/TCP’s exacerbated HOL blocking in developing regions. A multistreamed transport such as SCTP eliminates inter-object HOL blocking, enabling concurrent rendering. Evaluations using implementation of HTTP/SCTP in Apache and Firefox revealed that SCTP’s enhanced loss recovery enables faster page downloads than TCP in lossy, high latency paths. Also, concurrent rendering reduces pipelined objects’ response times, and the improvements are more promising for such objects as progressive images. These results motivated us to propose a low cost HTTP/SCTP deployment solution for developing regions. The authors hope that this work raises interest in using HTTP/SCTP to enhance browsing experience for web users in developing regions, and welcome further research and collaboration along these lines.

7. DISCLAIMER
The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U. S. Government.

8. REFERENCES
[10] HTTP-NG working group. www.w3.org/Protocols/HTTP-NG