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 [Natarajan 2006], 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. Since [Natarajan 2006], we have worked on adapting the Apache and Firefox implementations to handle HTTP 1.1 persistent, pipelined transfers over SCTP streams. Initial emulation results over high latency paths reveal that HTTP over multistreamed SCTP 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 [Movies]. The promising results have motivated us to propose a low cost, easily realizable, gateway based HTTP over SCTP deployment solution to enhance web 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≥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*, is due to the fact that 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 data arriving on other streams. The Stream Control Transmission Protocol (SCTP) [RFC4960] is a standardized reliable transport protocol which provides transport layer 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 pieces of embedded web objects compared to traditional *sequential rendering* in HTTP/TCP, especially in scenarios where HTTP/TCP suffers from exacerbated HOL blocking. Interestingly, we discovered that a multistreamed transport enables concurrent rendering even under no loss conditions [Natarajan 2007].

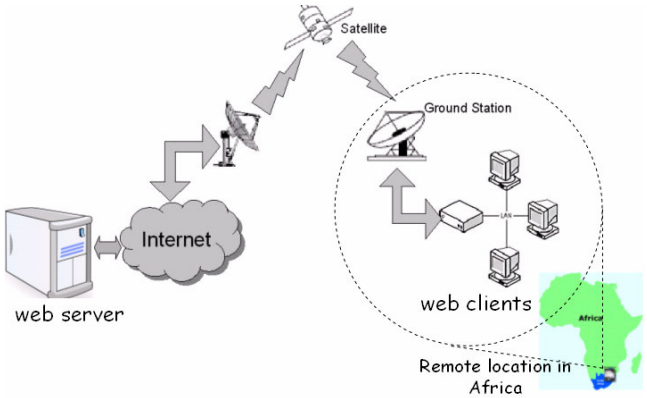

Figure 1: Internet Connectivity via VSAT Link

While most web users in developed nations experience end-to-end delays in the order of 10's of milliseconds, a large and growing portion of WWW users in developing nations experience much higher delays, ranging from 100's of milliseconds to a few seconds. Such high delays transpire from low bandwidth and/or high propagation delay links, such as VSAT/3G/GPRS links. For a multitude of factors, VSAT solutions (Figure 1) are the most cost-effective and efficient method of providing Internet connectivity for commercial customers, governments and consumers in developing nations, and other areas where a land-based infrastructure does not exist. The successful deployment of

* Prepared through collaborative participation in the Communication and Networks Consortium sponsored by the US Army Research Lab under Collaborative Tech Alliance Program, Coop Agreement DAAD19-01-2-0011. The US Gov't is authorized to reproduce and distribute reprints for Gov't purposes notwithstanding any copyright notation thereon.

Supported by the University Research Program, Cisco Systems, Inc.

VSAT systems and services in more than 120 countries provides communities with access to information, knowledge, education and business opportunities, and is crucial to socio-economic development [Rahman 2002].

The propagation delay from ground station to geostationary satellite to ground station is ~280ms [RFC3135]. Therefore, a VSAT link in an end-to-end path increases the RTT by at least 560ms. The bandwidth-limited VSAT link is most likely the bottleneck in the transmission path. Any resulting queuing and transmission delays further increase the RTT. Also, the delay caused by shared channel access in VSAT links can increase the RTT on the order of a few seconds [RFC3135].

Apart from propagation delays, sub-optimal traffic routing increases web response times for users in developing nations [Baggaley 2007, Cottrell 2006]. 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 RTTs as high as 2.5 seconds [PingERb].

In this work, we focus on concurrent rendering's ability to enable visually perceivable response time improvements in high latency browsing environments. According to [Akamai 2006], 4 seconds is the highest acceptable response time for retail web pages. Web page download times above 4 seconds interrupt the user experience, causing the user to leave the site or system. While web users over a high latency path 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 in high latency environments. Section 3 briefly describes our HTTP over SCTP streams design and implementation. Section 4 presents our emulation results and observations. Section 5 proposes a realistic, low cost, gradual deployment solution that enables web users in developing regions to benefit from concurrent rendering. Section 6 discusses solutions proposed thus far to reduce web response times. Finally, Section 7 summarizes the work.

2. HOL Blocking in High Latency Networks

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

obj_i = object i, $0 \le i \le N$. obj_0 denotes index.html, $obj_{1..N}$ denote the embedded objects in index.html.

req_i = time when the web client generates the HTTP GET request for obj_i, and writes the request to the transport layer.

$obj_i^k = k^{th}$ piece of obj_i, $0 \le k \le M$; obj_i^0 denotes the response header, and $obj_i^{1..M}$ denote the different pieces of obj_i. Note that M depends on the size of obj_i. In our emulations, we assume all objects are the same size (M).

rsp_i^k = time when transport delivers obj_i^k to the web client.

ren_i^k = time when web client renders obj_i^k on user's monitor.

$proc_i^k = (ren_i^k - rsp_i^k)$ denotes the web client's processing time for obj_i^k, such as decoding and rendering a JPEG image.

In TCP's sequential rendering, if obj_i^k is lost and recovered after x time units, pieces of obj_j $(j > i)$ could be HOL blocked for x time units. Assuming the web client is currently rendering obj_i^{k-1}, if $(x < proc_i^{k-1})$, this instance of HOL blocking does not affect response time for obj_{j+1}. Otherwise, the HOL blocking increases obj_{i+1}'s response time by $(x - proc_i^{k-1})$ 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, object sizes also influence the degree of HOL blocking in a pipelined transfer. As object size increases, the probability that a piece of the object is lost also increases. Hence, a larger object is more likely to block delivery of subsequent objects than a smaller object would.

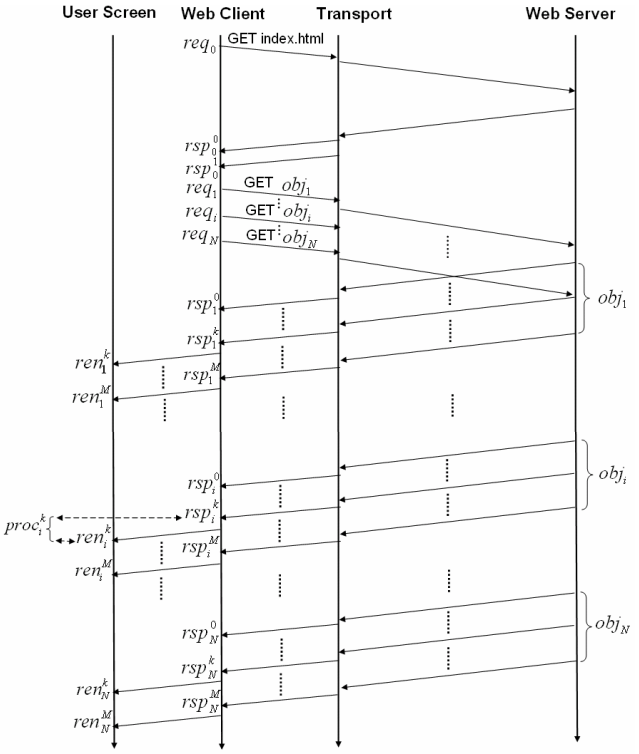

Figure 2: HTTP 1.1 Persistent, Pipelined Transfer Timeline

3. HTTP OVER MULTISTREAMED SCTP

SCTP was originally designed within the IETF SIGTRAN working group to address the shortcomings of TCP for telephony signaling over IP networks. SCTP has evolved into a general purpose IETF transport protocol. More than 25 SCTP implementations currently exist, including kernel implementations for most UNIX-like operating systems, user-space implementation for Windows, and implementations on proprietary platforms for Cisco, Nokia, Siemens, and other vendors. Nine interoperability workshops over the past seven years have fine-tuned these implementations [sctp.org].

We designed HTTP/SCTP streams and initially implemented the same for non-pipelined (HTTP 1.0) transactions in the open source Apache web server and Firefox browser [Natarajan 2006].

As mentioned in Section 1, a multistreamed transport's primary contribution to improve web response times is concurrent rending which enables a browser to interleave the rendering of *multiple* pipelined objects in parallel. Since [Natarajan 2006], we have modified Firefox to handle concurrent rendering over SCTP streams. During this process, we discovered that HTTP/SCTP streams on FreeBSD enables concurrent rendering even under no loss conditions. Details of this effort can be found in [Natarajan 2007].

4. EVALUATION

The following browsing environments, typical of current high latency Internet connections [PingERb] are considered for evaluation. Results for other high latency environments such as High Speed Download Packet Access (HSDPA) links are available in [Natarajan 2007].

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.

4.1 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 [Rizzo 1997] 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 browsing environments we consider [PingERb].

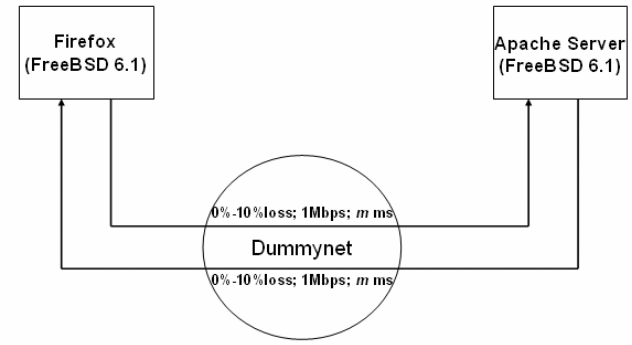

Figure 3: Emulation Setup

To best comprehend the behavior of concurrent rendering, we compare an HTTP 1.1 persistent, pipelined transfer over a single TCP connection with an identical transfer over a single multistreamed SCTP association. RFC2616 recommends web browsers to open a maximum of 2 transport connections to the same server/proxy. Splitting a pipelined transfer over 2 transport connections reduces the number of pipelined responses transmitted per connection. Therefore, our emulations consider varying number of objects per pipelined transfer (see below).

We have found disparities between the RFC2616 recommended number of open transport connections to a server, 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] also applies to an application using multiple SCTP associations. Additionally, web transfers over multiple transport connections reduce the number of "in flight" Transport Protocol Data Units (TPDUs) per connection. Insufficient number of duplicate acks to trigger fast retransmits increases the chances of timeout based loss recoveries in such connections [Balakrishnan 1998]. TPDU losses lower congestion window (cwnd) which further decreases the number of "in flight" TPDUs, contributing to more timeout recoveries. Timeout recoveries degrade response times, especially in low bandwidth/high latency environments found in developing regions. Therefore, improving loss recovery by increasing the number of in flight TPDUs per transport connection could be crucial to improve response times in lossy low bandwidth/high latency environments.

At both client and server nodes, we assume that the transport layer send and receive buffers are not the bottlenecks for throughput; they are large enough to hold all data of pipelined transfer. Several web workload characterization studies reveal that the file size distribution on web servers and the transferred file size distribution are heavy-tailed (Pareto) [Williams 2005]. Every pipelined transfer in the emulations contains equal sized objects of 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.

Page download time has been a popular metric for web response times. However, page download times are unaffected by the concurrency level (number of streams) of a multistreamed web transport, and do not identify improvements such as reduced HOL blocking. Therefore the evaluations also consider individual object's response times.

4.2 Page Rendering Time

A web page is completely rendered when Firefox draws the last embedded object in the pipelined transfer. The last piece of data always belongs to the last pipelined object in sequential rendering, but could be the last piece of *any* pipelined object in concurrent rendering. In both schemes, rendering the *last* piece of object depends on the throughput of the underlying transport connection.

Page rendering time is considered the time from when the browser sends the first GET request (index.html), to the time when the complete page is rendered on the screen. Using terminology defined in Section 2,

$$\textit{Page rendering time (T)} = (ren_N^M - req_0)$$

Our initial hypotheses about SCTP and TCP's page rendering times were the following:

(i) Both SCTP and TCP have similar values for initial congestion window (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 during no losses.

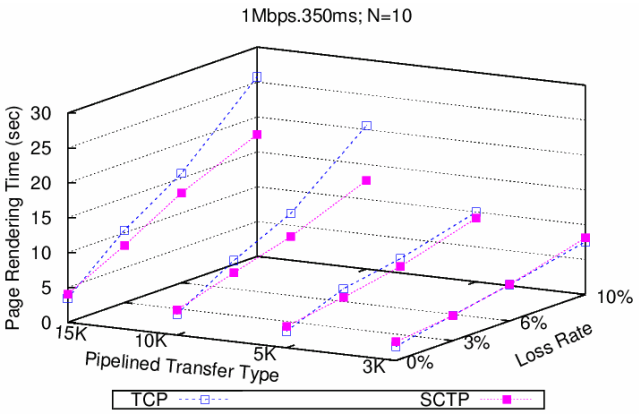

Figure 4: Page Rendering Times (1Mbps.350ms; N=10)

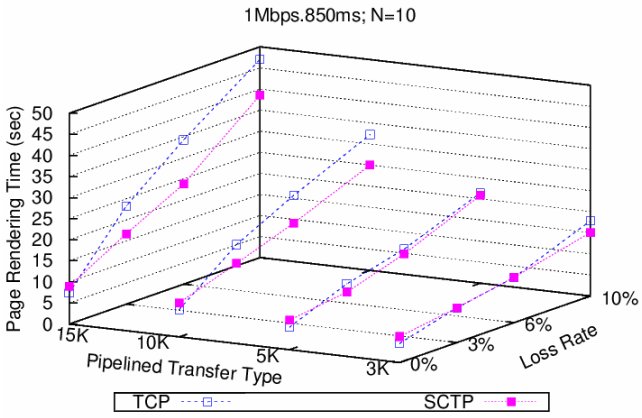

Figure 5: Page Rendering Times (1Mbps.850ms; N=10)

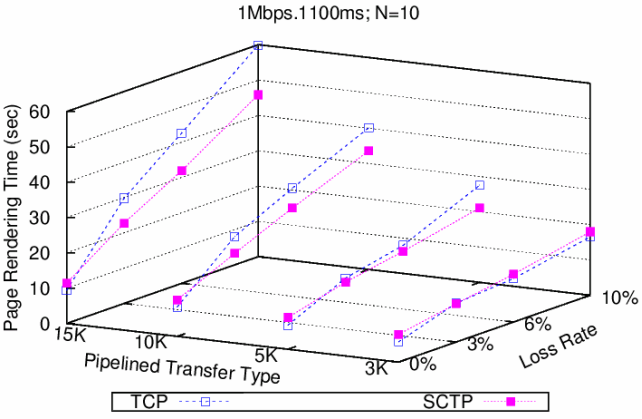

Figure 6: Page Rendering Times (1Mbps.1100ms; N=10)

(ii) Both SCTP and TCP employ selective acknowledgements (SACKs). Unlike TCP whose SACK info is limited by the space available for TCP options, the size of SCTP's SACK chunk is limited by the path MTU and contains more accurate 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 [Caro 2006]. Therefore, as loss rates increase, we expected the enhanced loss recovery features to help SCTP perform better than TCP.

Figures 4-6 show the page rendering times for N=10, averaged over 50 runs with 95% confidence level. Similar results for N=5 and 15 can be found in [Natarajan 2007]. Interestingly, in all 3

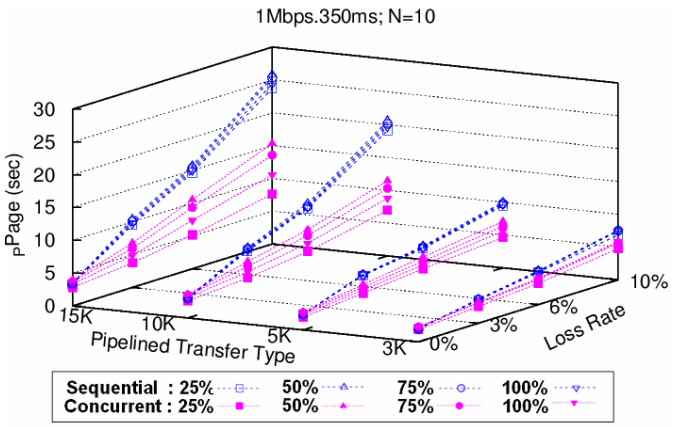

Figure 7: $_{P}$Page Values for 1Mbps.350ms; N=10

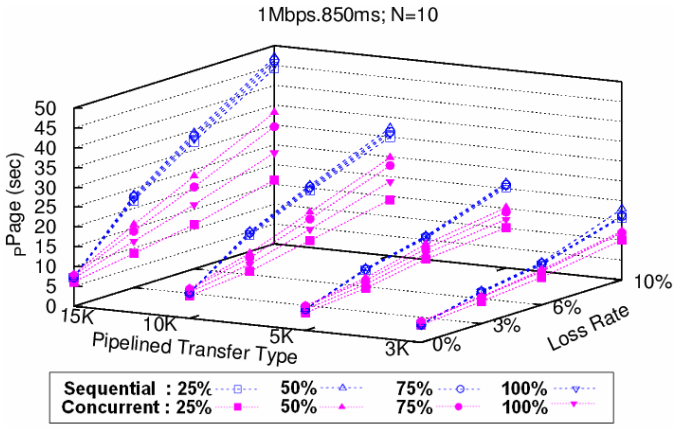

Figure 8: $_{P}$Page Values for 1Mbps.850ms; N=10

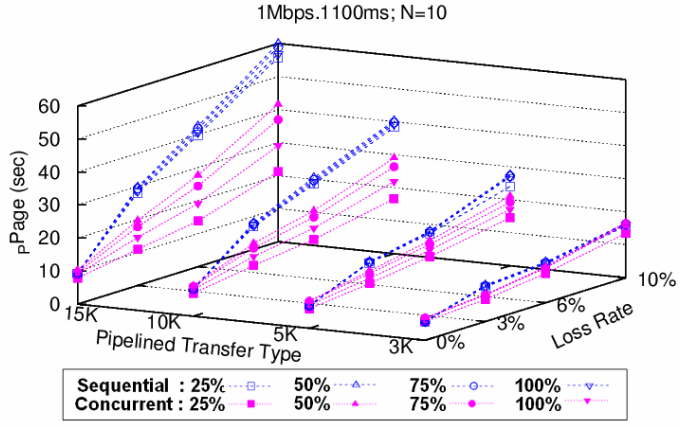

Figure 9: $_{P}$Page Values for 1Mbps.1100ms; N=10

graphs, the results for the no loss case, contradict (i), and TCP's rendering times are slightly (but not *perceivably*) 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

this difference to disappear when we compare the results with a TCP stack that implements ABC with L=1.

As loss rate increases, SCTP's enhanced loss recovery offsets the difference in SCTP vs. TCP cwnd evolution. SCTP begins to perform better, and the difference is more pronounced for transfers containing larger objects (10K and 15K). For the 1Mbps.1100ms case, the difference between SCTP and TCP page rendering times for 10K and 15K transfers is ~6 seconds for 3% loss, and as high as ~15 seconds for 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 improvements are increasingly *visually perceivable* as the end-to-end delay, loss rate, and pipelined transfer size increase.

4.3 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. An SCTP association with two streams provides twice as much concurrency as sequential rendering. A multistreamed association provides maximum concurrency for a pipelined transfer when the number of streams equals the number of objects in the transfer. Note that concurrent rendering remains unaffected by further increase in concurrency.

We study the pipelined objects' response time differences between the following two rendering schemes: (i) concurrent rendering with maximum concurrency, and (ii) sequential rendering. 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,

req_0 = time when browser sends HTTP GET request for index.html.

$({}_Pren_i - req_0)$ = time elapsed from the beginning of the page download (req_0) to the earliest time when at least P% of object i 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. We define ${}_PPage$ as the time elapsed from the beginning of page download to the earliest time when at least P% of *all* pipelined objects have been rendered on the screen, i.e.,

$${}_PPage = MAX\ [({}_Pren_i - req_0);\ 1 \le i \le N]$$

Figures 7- 9 show the ${}_{25\%}Page$, ${}_{50\%}Page$, ${}_{75\%}Page$ and ${}_{100\%}Page$ values for N=10, averaged over 50 runs with 95% confidence level. Results for N=5 and 15 can be found in [Natarajan 2007].

We first observe that, as expected, ${}_{100\%}Page$ values are equal to the corresponding transport's page rendering times (T). Also, the ${}_PPage$ times in concurrent rendering (shown by pink points in the graphs) are spread out vs. clustered together in sequential rendering (shown by blue points in the graphs). Concurrent rendering's dispersion in ${}_PPage$ values signifies the parallelism in the appearance of all N=10 pipelined objects.

Both sequential and concurrent rendering schemes' values are comparable at 0% loss. As noted before, our implementation of HTTP/SCTP on FreeBSD enables concurrent rendering even under no loss conditions, causing concurrent rendering's ${}_{25\%}Page$ to be slightly improved than sequential rendering's (discussed in detail in [Natarajan 2007]). As loss rate increases, the difference in two rendering schemes' ${}_PPage$ values increase. Interestingly, 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.

4.3.1 *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.

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) rendering schemes download the example web page over the 56Kbps.1080ms setup. The full page downloads were captured as movies, and are available online at [Movies].

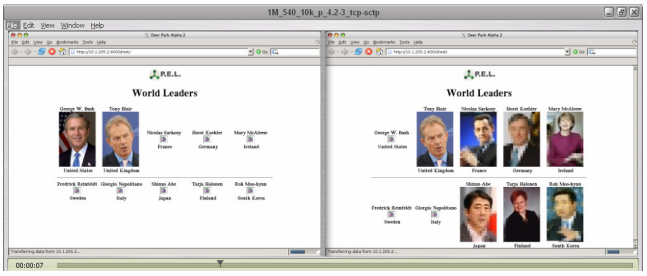

Figure 10a: Concurrent Rendering of Progressive Images (56Kbps.1080ms; 4.3% loss; t=7s)

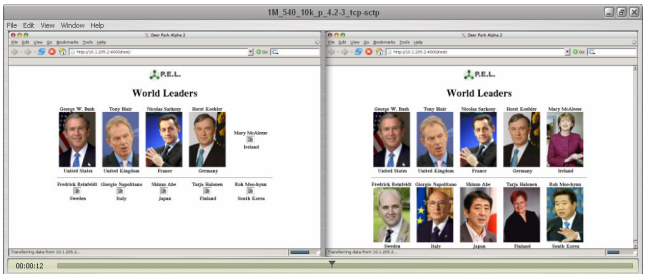

Figure 10b: Concurrent Rendering of Progressive Images (56Kbps.1080ms; 4.3% loss; t=12s)

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 images, at least 4 of which are of good quality. At time=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 ~23s. From 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 23s. In the sequential run, all 10 images appear on the screen only ~26s.

5. APPLICATION TO DEVELOPING REGIONS

While concurrent rendering's initial results promise better response times for users in high latency environments, it is impractical to expect all web servers to provide web over multistreamed SCTP in the immediate future, without which users in developing regions 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 provides better page rendering times than HTTP/TCP. This solution can also be extended to further enhance pipelined objects' response times. For example, the gateway could use batch image conversion software [Gimp] to convert embedded JPEG/PNG images to the corresponding progressive versions before forwarding them to the clients. The image conversion at the gateway takes on the order of milliseconds per image, and can improve the user's response times on the order of seconds.

6. 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 [Gettys 2002]) 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 [Gettys 1998] was one such session management protocol that was a product of the (now historic) HTTP-NG working group [HTTP-NG]. WebMUX proposed using a reliable transport protocol to provide web transfers with "streams" for transmitting independent objects. However, the WebMUX effort did not mature.

[Ford 2007] proposes the use of Structured Stream Transport (SST) for web transfers. SST functions similar to SCTP multistreaming by extending TCP to offer multiple streams over a TCP-friendly transport layer connection. [Ford 2007] focuses on design and development of SST, while this work analyzes user perceivable response time improvements with a multistreamed web transport.

Content Delivery Networks (CDNs) replicate web content across geographically distributed servers, and reduce response times for web users by redirecting requests to a server closest to the client. [Krishnamurthy 2001] confirms that CDNs reduce average web response times for web users along USA's east coast for static content. 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 in the last hop, as is the case for VSAT and 3G-HSDPA links and/or (ii) sub-optimal traffic routing that increases end-to-end path RTTs for Internet traffic on the order of seconds.

7. SUMMARY

In this work we examined exacerbated HOL blocking in HTTP/TCP for web clients in developing regions connected by high latency networks. A multistreamed transport such as SCTP delivers independent objects without HOL blocking, enabling concurrent rendering and improved response times at the browser. Using our implementation of HTTP/SCTP in Apache and Firefox, we showed that SCTP's enhanced loss recovery enables perceivable improvements to web page rendering times in high latency paths. Also, concurrent rendering of pipelined objects results in remarkable improvements to response times, and the improvements are more promising for objects such as progressive images. We also discussed a low cost solution for easier deployment of HTTP/SCTP in high latency networks.

The authors hope that our results raise interest for using HTTP/SCTP to improve Internet browsing experience for users in developing regions, and welcome further research and collaboration along these lines.

8. 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.

9. REFERENCES

[Akamai 2006] "Akamai and JupiterResearch Identify 4 Seconds as the New Threshold of Acceptability for Retail Web Page Response Times," Press Release, Akamai Technologies Inc, Nov 2006. URL: http://www.akamai.com/html/about/press/releases/2006/press_110606.html.

[Baggaley 2007] J. Baggaley, B. Batpurev, J. Klaas, "Technical Evaluation Report 61: The World-Wide Inaccessible Web, Part 2: Internet routes," International Review of Research in Open and Distance Learning, 8(2), 2007.

[Balakrishnan 1998] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, R. Katz, "TCP behavior of a busy Internet server: Analysis and Improvements," IEEE INFOCOM, San Francisco, Mar 1998.

[Caro 2006] A. Caro, P. Amer, R. Stewart, "Retransmission Policies for Multihomed Transport Protocols," Computer Communications, 29(10), Jun 2006.

[Cottrell 2006] L. Cottrell, A. Rehmatullah, J. Williams, A. Khan, "Internet Monitoring and Results for the Digital Divide," International ICFA Workshop on Grid Activities within Large Scale International Collaborations, Sinaia, Romania, Oct 2006.

[Ford 2007] B. Ford, "Structured Streams: A New Transport Abstraction," ACM SIGCOMM 2007, Kyoto, Japan, August 2007.

[Gettys 1998] J. Gettys, H. Nielsen, "The WebMUX Protocol," Internet Draft, August, 1998.

[Gettys 2002] J. Gettys, Email to End2end-interest Mailing List, October, 2002. www.postel.org/pipermail/end2end-interest/2002-October/002436.html.

[Gimp] The GNU Image Manipulation Program, URL: www.gimp.org

[HTTP-NG] HTTP-NG working group (historic). URL: www.w3.org/Protocols/HTTP-NG/

[Jurvansuu 2007] M. Jurvansuu, J. Prokkola, M. Hanski, P. Perala, "HSDPA Performance in Live Networks," IEEE International Conference on Communications (ICC) 2007, Glasgow, Scotland, Jun 2007.

[Krishnamurthy 2001] B. Krishnamurthy, C. Wills, Y. Zhang, "On the Use and Performance of Content Distribution Networks," ACM SIGCOMM Internet Measurement Workshop, California, USA, Nov 2001.

[Movies] HTTP/SCTP vs. HTTP/TCP Movies, URL: http://www.cis.udel.edu/~amer/PEL/leighton.movies/index.html

[Natarajan 2006] P. Natarajan, J. Iyengar, P. Amer, R. Stewart, "SCTP: An Innovative Transport Layer Protocol for the Web," 15th International conference on WWW, Edinburgh, May 2006.

[Natarajan 2007] P. Natarajan, P. Amer, R. Stewart, "The Case for Multistreamed Web Transport in High Latency Networks," Technical Report #2007-342, Computer & Information Sciences Dept, Univ. of Delaware, Oct 2007.

[PingERa] The PingER Project, URL: http://www-iepm.slac.stanford.edu/pinger/

[PingERb] PingER Detail Reports, URL: http://www-wanmon.slac.stanford.edu/cgi-wrap/pingtable.pl

[Rahman 2002] S. Rahman, M. Pipattanasomporn, "Alternate Technologies for Telecommunications and Internet Access in Remote Locations," 3rd Mediterranean Conference and Exhibition on Power Generation, Transmission, Distribution and Energy Conversion, Greece, Nov 2002.

[Rizzo 1997] L. Rizzo, "Dummynet: A simple approach to the evaluation of network protocols," ACM Computer Communications Review, 27(1):31-41, Jan 1997.

[Williams 2005] A. Williams, M. Arlitt, C. Williamson, K. Barker, "Web workload Characterization: Ten years later," Book Chapter in "Web Content Delivery," Springer, 2005. ISBN: 0-387-24356-9.