Measuring and Evaluating TCP Splitting for Cloud Services

Abhinav Pathak[1], Y. Angela Wang[2], Cheng Huang[3],
Albert Greenberg[3], Y. Charlie Hu[1], Randy Kern[3], Jin Li[3], and Keith W. Ross[2]

[1] Purdue University
[2] Polytechnic Institute of NYU, New York
[3] Microsoft Corporation, Redmond

Abstract. In this paper, we examine the benefits of split-TCP proxies, deployed in an operational world-wide network, for accelerating cloud services. We consider a fraction of a network consisting of a large number of *satellite* datacenters, which host split-TCP proxies, and a smaller number of *mega* datacenters, which ultimately perform computation or provide storage. Using web search as an exemplary case study, our detailed measurements reveal that a vanilla TCP splitting solution deployed at the satellite DCs reduces the 95^{th} percentile of latency by as much as 43% when compared to serving queries directly from the mega DCs. Through careful dissection of the measurement results, we characterize how individual components, including proxy stacks, network protocols, packet losses and network load, can impact the latency. Finally, we shed light on further optimizations that can fully realize the potential of the TCP splitting solution.

1 Introduction

Cloud Services are delivered with large pools of computational or storage resources that are concentrated in *mega* datacenters, being built only in a hand full of remote locations world-wide. The continued growth of Cloud Services, however, critically depends on providing a level of responsiveness comparable with what can be obtained directly from dedicated on-site infrastructures. A key challenge here is to make remote infrastructures appear to end-users as if they were nearby.

Split-TCP proxies [1][2] can be very effective in improving the responsiveness of Cloud Services. In particular, Cloud Service providers can deploy *satellite* datacenters – within or outside of their own networks – close to the end-users. These satellite DCs host split-TCP proxies, which maintain persistent connections over long-haul links to the mega DCs that ultimately perform computation or provide storage. Through bypassing TCP *slow start* and avoiding a number of round trips on the long-haul links, this architecture reduces response time very effectively. Many Cloud Services, whose computation or storage cannot be easily or cost-effectively geo-distributed and thus have to remain in the mega DCs, can benefit from this architecture, including Internet search, collaborative online editing, concurrent social gaming, cloud-based storage, and so on. Although there are existing deployments of split-TCP proxies in commercial systems (e.g., [3][4]), there are very few published studies on performance gains based

A. Krishnamurthy and B. Plattner (Eds.): PAM 2010, LNCS 6032, pp. 41–50, 2010.

on real-world Internet measurements. Furthermore, to our knowledge, there is no thorough study that dissects each component of a TCP splitting solution, with the aim at identifying further optimizations and fully realizing its potential.

In this paper, we deploy an experimental TCP splitting solution on a fraction of a network of satellite DCs hosted by Microsoft's global distribution and cloud service network. We conduct detailed measurements to quantify the gain experienced by real-world end-users. Using web search as an exemplary case study, we show that, compared to directly sending queries to the mega DCs, a vanilla TCP splitting solution can reduce the 95^{th} percentile latency by 43%[1]. Through careful dissection of the measurement results, we characterize how individual components – including proxy stacks, network protocols, packet losses and network load – can impact the latency. Finally, we shed light on further optimizations that can fully realize the potential of the TCP splitting solution.

2 A Web Search Case Study

Web search is one of the most important and popular cloud services. Clearly, the relevance of search result is critical to the success of the service. In addition, the speed of search response is essential. Delay of an extra half a second can affect user satisfaction and cause a significant drop in traffic [7].

2.1 Search Response: Empirical Results

The amount of data in a search response is typically very small. To measure its size and time, we identified about 200,000 common search terms from anonymized reports from real-world users using 'MSN Toolbar'. For each search term, we issued a query to obtain a uncompressed HTML result page, against a popular Internet search engine (the search engine name is anonymized). We issued all the queries from a single client located on a university campus network. We measured the response size and the response time, that is, the time elapsed from TCP SYN is sent until the last packet of the HTML result page is received. We also extracted the time – taken within the search datacenter to complete the actual search and construct the result – as reported on the resultant HTML page.

Figure 1 plots the CDF response size of 200K search queries. We see that the size of a typical uncompressed search response is 20-40KBytes, sometimes exceeding 50KBytes. With a TCP Maximum Segment Size (MSS) of about 1500 bytes, this corresponds to 15 to 27 TCP data packets utilizing 4 TCP windows of data transfer (as suggested by RFC 3390 [8]). Figure 2 plots the CDF of response time of 200K search queries as observed by the client and the CDF of search time within the datacenter (denoted "Search Time") as reported on the result page. From the figure, we see that a typical response takes between 0.6 and 1.0 second. The RTT between the client and the datacenter during the measurement period was around 100 milliseconds. We remark that our measurement

[1] In this paper, we focus on optimizing split-TCP solutions under given satellite DCs. We cover an orthogonal and equally important issue – the choice of satellite DC locations – in separate studies [5][6].

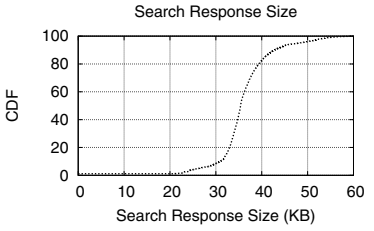

Fig. 1. CDF of response sizes of 200K search queries from a popular search engine

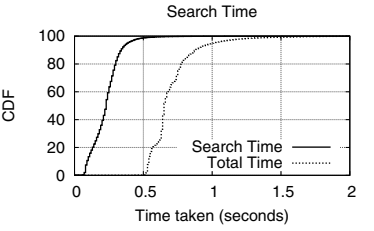

Fig. 2. CDF of response time of 200K search queries by popular search engine for search reply

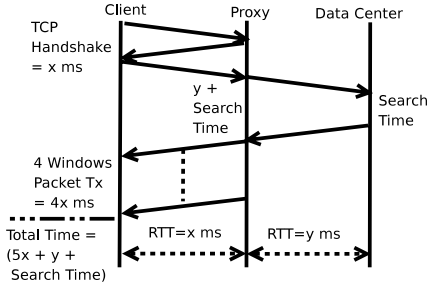

Fig. 3. TCP packet exchange diagram between an HTTP client and a search server with a proxy between them

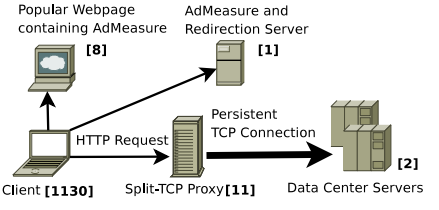

Fig. 4. TCP splitting platform - The [number] beside each component represents the number(s) of that component in our measurement system

client was connected from a well-provisioned network. As we will see later, the response time can be a lot worse for clients in the wild. From the figure, we see that the search time within the datacenter ranges almost uniformly between 50 and 400 msec. We also note that 5.24% of the search queries took one second or more to finish. The 95th percentile of the response time was roughly one second.

2.2 Simple Model for Response Latency

The total time for a query and response is composed of the time to send the query, the time to construct the result page and the time to transfer the page (Other factors that influence the user perceived time includes page rendering speed, speed of scripts execution on browser, etc. While these are important factors, we only study the network part of latency in this paper). The client first establishes a TCP connection with the datacenter server through a three-way TCP handshake. The client then sends an HTTP request to the server, which includes the search query. The sever performs the search, incurring $search\_time$, and then ships the result page to the client. We assume that four TCP windows (as discussed earlier) are required to transfer the result page to the client when there is no packet loss. The total time taken in this case is $(5RTT + search\_time)$.

Now, consider the potential improvement of TCP splitting, where a proxy is inserted, close to the client, between the client and the datacenter, as shown in Figure 3. In such a design, the proxy maintains a persistent TCP connection with the data center server, where the TCP window size is large, compared to the amount of data in individual search result pages. A client establishes a TCP connection and sends a search query to the proxy. The proxy forwards the query to the datacenter server over the persistent connection with a large TCP window (We ignore the CPU processing overhead incurred at proxies throughout this paper as this overhead was estimated to be as low as 3.2ms in userspace split TCP implementation and 0.1ms in kernel level split TCP implementation [2]). The datacenter server processes the search query, incurring $search\_time$, and transmits the resulting page back to the proxy within one round trip (given the large TCP window). The proxy then transfers the data to the client, which goes through a TCP slow-start phase and takes several round trips.

The total time taken in this case is $(5x + y + search\_time)$, where x is the RTT between the client and the proxy and y is the RTT between the proxy and the datacenter. Comparing this with the no-proxy case, we see that TCP splitting can potentially reduce the response time by $(5RTT - (5x + y))$. When $x + y \approx RTT$ (i.e., the proxy detour overhead is negligible), this reduction becomes $4(RTT - x)$; when further $x << RTT$, i.e., the client-proxy distance is small when compared to the proxy-datacenter distance, this reduction becomes approximately $4RTT$, which can be quite substantial for interactive applications.

3 Experimental TCP Splitting System

To understand the gain of TCP splitting on search queries in the wild, as well as to characterize individual components in a TCP splitting design, we've implemented an experimental TCP splitting system, deployed it in a global distribution and cloud service network, and characterized the system with search traffic from real-world users. Our experimental system has two major components: a client measurement platform and a TCP splitting platform.

3.1 Measurement System

Client Measurement Platform: Our goal is to measure query latencies for real clients in the wild - with and without split-TCP proxies. To this end, we exploit AdMeasure, a measurement platform we recently developed [6]. In a nutshell, AdMeasure deploys a Flash object (implemented in 300 LOC in ActionScript) on multiple popular web pages. When a client visits any one of these web pages (as shown in Figure 4), the AdMeasure Flash object is loaded into the client at the end of the web page (so as not to affect user-perceived page load time). The Flash object retrieves a workload list from a central AdMeasure server, performs pre-configured Internet measurements such as issuing search queries to the IPs contained in the workload list, and submits results back to the AdMeasure server.

In our TCP splitting experiments, the AdMeasure server uses the client's geographic location (from its IP address) and instructs the client to issue search queries to the closest proxy (This step incurs overhead for each query. It is a simplified implementation

and should be replaced by a DNS server in a production system, where the DNS resolution overhead is amortized over many queries and thus minimal). The proxy with minimum geographic distance is a reasonable approximation to the proxy with minimum RTT, as shown in [9]. For simplicity, we use a fixed search query term "Barack Obama" with/without proxies. We verify that the repetition of the same query term experiences similar "search time" in the mega datacenters - i.e., there is no caching of search results at the datacenter when same queries are issued repeatedly. We deploy the AdMeasure Flash object on multiple popular partner websites, including the front page for Microsoft Research, the front page for Polytechnic Inst. of NYU, the front pages of three small online gaming websites, as well as a few personal homepages.

TCP Splitting Platform: Our TCP splitting platform consists of two parts: split-TCP proxies and mega datacenters. We deploy split-TCP proxy (about 2K LOC in C++) in a fraction of the satellite DCs of Microsoft's global distribution and cloud service network (11 locations worldwide - 6 in US, 3 in Europe and 2 in Asia). We choose 2 Live Search mega DCs (both in US). Each proxy forwards search queries to the closer (in terms of RTT) Live Search datacenter. The proxy does not cache the results of any search query. The proxy relays the queries on behalf of clients over a persistent HTTP connection to the datacenter. It stores statistics like response time, content length, query id, etc. In addition, packet traces in form of tcpdumps are recorded.

Through AdMeasure, each client is instructed to issue 6 back-to-back queries to the closest of the 11 proxies, which forwards to the closer of the two datacenters; and each query starts a new TCP connection to the proxy. We ignore the first two queries, which are meant to warm up the TCP transmission window between the datacenter and the proxy. This is to emulate production environments, where many queries and responses are multiplexed over the same datacenter-proxy connection. To understand the degree to which TCP splitting helps, as a baseline, each client also issues six queries directly to the datacenter.

3.2 Measurement Results

Through AdMeasure, we collected one week's worth of data consisting of 5,584 search queries through proxies from 1,130 unique clients out of which 952 were located in North America (covering 193 cities). The bias in clients' location originates from the fact that the websites, where AdMeasure was deployed, were popular mostly in North America. Using one week's worth of data, we now report our experimental results in this subsection. Since the current deployment of AdMeasure attracts significantly more users from North America than other continents, we report only clients originating from North America.

Latency Model Validation: We first validate whether the simple model described in Figure 3 indeed holds true with the real clients. We separate out the traces with packet loss in either proxy to client or datacenter to proxy communication (traces with loss will be re-visited later). We identify packet loss from the proxy-side tcpdump outputs. For simplicity, we assume that retransmission implies data packet loss. ACK loss is not easy to identify, but turns out *not* to be rare. Here, we apply a simple heuristic to infer ACK loss – the sequence number gap between any two consecutive ACKs is calculated; if the

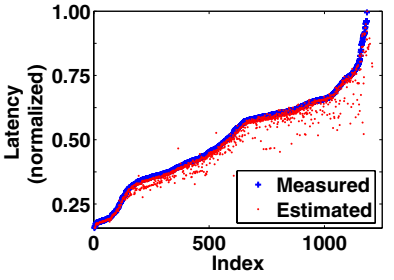

Fig. 5. Latency Model Validation

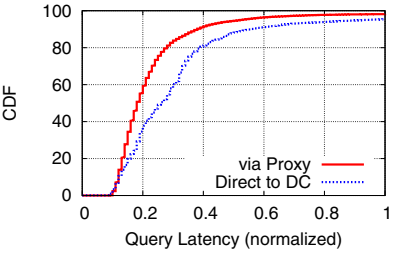

Fig. 6. Gain of TCP Splitting

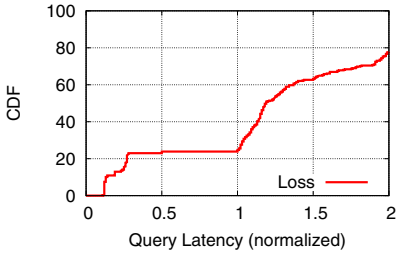

Fig. 7. Impact of Packet Loss

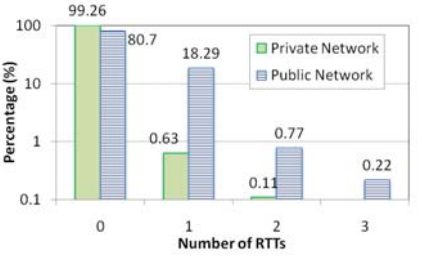

Fig. 8. Time (*RTT) between first and last response packet at proxy from datacenter

gap is bigger than two MSS (taking into account delayed acknowledgment [10]), we assume there is an ACK loss (this simple heuristic might over-estimate, but is nevertheless conservative for weeding out connections with loss).

Considering all traces without data packet/ACK loss, we now calculate an estimated latency as: 6 times RTT[2] between client and proxy ($= 6 * x$ - notation follows from Figure 3) + 1 RTT between proxy and datacenter ($= y$) + datacenter search time ($= search\_time$). Here, ($y + search\_time$) is obtained directly from tcpdump, as the time from when the proxy forwards the query to the datacenter until it gets the first response packet. We also obtain the true total latency from tcpdump, as the time from when the proxy receives the SYN until the final ACK from the client.

Figure 5 (best viewed in color) plots the estimated and true (normalized) latency of minimum of the six search queries per client visit (a few clients visited multiple times) sorted by measured latency. Using a linear scale, we normalized (and hence anonymized) all the real latencies with a constant large latency value throughout the paper. It is clear that, without packet/ACK loss, the simple model nearly approximates the true latency. Note that towards right in the figure, where the total latency is large, the RTTs between the clients and the proxies are also large. Larger RTTs show more variations, which tend to have a larger impact on inaccuracy.

[2] As for "Barack Obama" Query, response size is 55KB which is 5 TCP windows of data, implying, 1 RTT for TCP Handshake + 5 window packet transfer (5*RTT).

Table 1. TCP window comparison, for different OS's

	Linux	Win2003	Win2008
Window Size in Bytes			
1^{st}	2,920	2,520	2,920
2^{nd}	4,380	3,780	5,840
3^{rd}	5,840	4,980	11,680
4^{th}	8,760	7,560	23,360
5^{th}	11,680	11,340	
6^{th}	16,060	16,380	
# of Windows for 50KB+ data			
	7	7	5

Impact of Proxy OS: In the course of our experiment, we found that the operating systems on the proxies have a large impact on the latency, as they exhibit very different TCP window behavior. Table 1 compares the transmission window for Linux kernel 2.6.20, Windows Server 2003, and Windows Server 2008, using real data collected from our proxies. As we can see, both Windows Server 2003 and Linux show a similar window growth rate of about 1.5, whereas Windows Server 2008 shows a growth rate of 2. We believe the difference is because Windows Server 2008 implements Appropriate Byte Counting (ABC) [11], while Linux and Windows Server 2003 do *not*. ABC increases the TCP congestion window by the number of bytes acknowledged, compared to by the number of acknowledgements conventionally. Under delayed ACK (most common), the difference is exactly 2 vs. 1.5. Using Windows Server 2008 can immediately reduce the total latency by two round trips between the client and the proxy. Hence, all results reported in the paper use proxies hosted on Windows Server 2008 machines.

How Much Does TCP Splitting Help? Recall that each client issues six queries through the proxy and the first two responses are used to warm up the TCP transmission window between the datacenter and the proxy. The performance of the remaining four queries should reflect latencies that end-users will experience.

We now include all the cases - with and without data packet or ACK loss and present the main finding of this section: a comparison of the end-to-end latency with and without TCP splitting. Figure 6 plots the CDF of search latency with and without TCP splitting. We see that at 95^{th} percentile, TCP splitting reduces latency from 0.93 to 0.53 (both values are normalized) – a savings of 43%!

Impact of Packet Loss: Loss can occur either between the datacenters and the proxies, or between the proxies and the clients. But, as we will see next, the latter case is much more common. Indeed, from the packet traces, we observe that 7% of the TCP sessions between the proxies and the clients have at least one packet retransmission. To examine the impact of loss, Figure 7 plots the CDF of the response time for these 7% queries. From the figure, we observe that (1) when compared to all the queries (the "via proxy" curve in Figure 6), loss significantly impacts the latency – the (normalized) response time of 76% queries is greater than 1.0; and (2) some of the queries are heavily affected - the (normalized) response time of 22% queries with packet loss is greater than 2.0.

An implication of the finding is - if loss in the last mile can be handled effectively, the latency of the 7% queries can be drastically reduced.

Latency in Hauling Data from Datacenter: In our experimental deployment, each proxy maintains a persistent TCP connection with the datacenter. If packet loss occurs between the proxy and the datacenter, additional round trips are required to transmit a response. Using the last four of the total six queries per test, we now examine whether the datacenter can always transmit the entire response to the proxies in one transmission window. In particular, we examine the time gap between the proxy receiving the first and last packet from the datacenter. Ideally, this time should be close to 0, if all packets arrive in a single window. Figure 8 (*private-network* bars) shows that this is typically the case for the proxies within Microsoft's global distribution network. For comparison purposes, we have also deployed a split-TCP proxy inside the Abilene network at Purdue University (*public-network* bars in figure 8). For this proxy, in sharp contrast, about 20% of the cases take one RTT (round trip time between this proxy and datacenter), indicating that at least one packet loss has occurred. An implication of this finding is - when Cloud Service providers start to deploy satellite DCs beyond their private networks, they are more likely to encounter the so-called "middle-mile" problem [3]. In that case, a customized FEC-based low latency reliable protocol (e.g.,[12]) between the proxy and the datacenter should be beneficial.

Stress Testing: During our measurements, we were able to attract over 1000 clients in one week's time frame. This rate does not give us the opportunity to measure the performance of our system under load. Specifically, we wanted to measure the response time between the proxy and the datacenter under load. As nearby queries go through the same connection between the proxy and the datacenter, they would suffer the same fate , i.e., if one query experiences a loss, the queries succeeding it would also suffer due to TCP semantics.

To stress-test our split-TCP platform, we conducted the following experiment in all the 11 proxy locations. Due to the lack of high client arrival rate, we made each proxy issue back-to-back queries to itself destined to the datacenter. Each query fetches a single image of about 50KB size from a web server in the datacenter, over a single persistent HTTP connection. We choose to fetch a single static image since this would incur negligible time at web server and we would have only the network under the microscope. The web server was configured with unlimited HTTP requests over the persistent HTTP connection. The query rate was varied from 1 request/sec to 1000 request/sec. At each rate, we dispatched 10000 queries and waited for all the responses to be received. For every query, we measured the response time. Since we were fetching a static image from the web server, the latency of the web server itself should be negligible. Over the persistent HTTP connection with a large TCP window, the response time should be a single RTT.

Figure 9 plots the results of stress testing for 8 of the 11 proxy locations (the rest 3 similar). For every location, we plot one bar for each request rate indicating the percent of requests that took 2 or more RTT to complete. For example, for the proxy in Amsterdam, at 1 request/sec, 0.02% of the requests took 2 or more RTT. We issued a total of 10,000 requests (1 per second). This means that 2 out of the 10,000 requests took

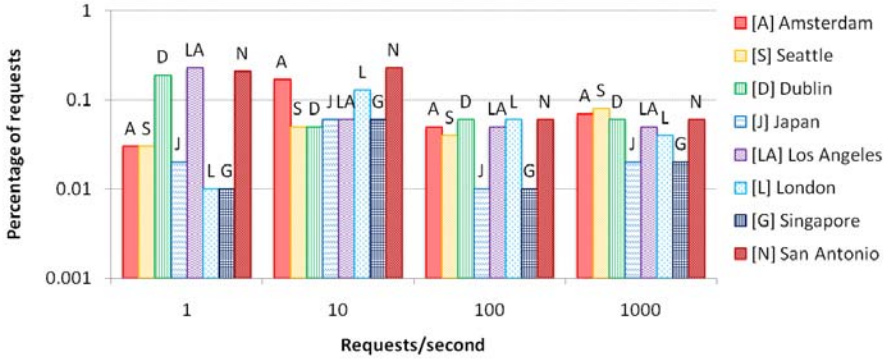

Fig. 9. Stress Test: Percentage of requests that took more 2 or more RTTs to complete

more than 2 RTT. Those were in fact the first two requests that warmed up the TCP connection. Even at a high request rate, less than 0.1% of the queries incur more than 2 RTTs. This experiment shows that, in the private network of satellite datacenters, even high load can be readily handled by persistent HTTP connections.

4 Related Work

Proposals for using persistent-connection HTTP and split-TCP proxy to improve web transfer performance can be dated back at least to the mid 90's. Early important work includes that of Padmanaban [13] and Mogul [14]. Proxies can provide additional benefit through clever techniques, such as using static content to open up TCP window [4]. Furthermore, proxies can provide benefit through adaptations of piggyback mechanisms [15]. Authors in [1][2] evaluate performance of split-TCP proxies using a very limited set of clients. Through emulation, [1] evaluated socket-level TCP splice using a single client and various latency/loss rate. The focus was to estimate the number of CPU cycles that a proxy spends processing requests. [2] estimated the latency penalties incurred by split-TCP proxies. Authors estimated that a kernel level split-TCP implementation incurs only 0.1ms. However, none of the studies were deployed and evaluated using a production environment and through a large number of real-world end-users. Moreover, none of studies dissects the splitting TCP solution from as many aspects as we do, nor do they outline the directions for further optimizations.

5 Conclusion and Future Work

In this paper, we investigate the benefits and optimizations of TCP splitting for accelerating Cloud Services. Using web search as an exemplary case study and through an experimental system deployed in a production environment, we show that TCP splitting can indeed reduce the response time of Cloud Services significantly. We also identify

a number of directions for further optimizations in order to achieve the full benefit of TCP splitting.

During our experimental deployment, we observe that packet loss is rather common between the end-users and the proxies, even though the proxies are deployed in a well-provisioned and well-connected production network. This is a bit surprising, but yet consistent with observations from other production networks [4]. As an ongoing work, such reality prompts us to pursue TCP stack modifications on the proxy so as to more effectively handle packet loss and improve the latency performance.

Furthermore, along with optimizing each component in the TCP splitting system, expanding the presence of the global distribution network (and thus proxies) will also help. The holy grail question being – how many locations will be sufficient and where should these locations be? We are developing new methodologies [6] and conducting large scale studies in order to answer this question conclusively.

References

1. Ibm, R.U., Rosu, D.: An Evaluation of TCP Splice Benefits in Web Proxy Servers. In: WWW. ACM Press, New York (2002)
2. Maltz, D.A., Bhagwat, P.: TCP Splicing for Application Layer Proxy Performance. Technical report, IBM Research Report 21139 (Computer Science/Mathematics) (1998)
3. Akamai: Akamai's EdgePlatform for Application Acceleration. Akamai, Inc. (2007)
4. Tariq, M., Zeitoun, A., Valancius, V., Feamster, N., Ammar, M.: Answering What-If Deployment and Configuration Questions with WISE. In: ACM SIGCOMM (August 2008)
5. Huang, C., Wang, Y.A., Li, J., Ross, K.W.: Measuring and Evaluating Large-Scale CDNs. MSR Technical Report MSR-TR-2008-106 (2008)
6. Wang, Y.A., Huang, C., Li, J., Ross, K.W.: Measuring Network Performance for Cloud Services with AdMeasure (2009) (Submitted)
7. Mayer, M.: Web 2.0, `http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html`
8. Allman, M., Floyd, S., Partridge, C.: Increasing TCP's Initial Window. RFC 3390 (October 2002)
9. Krishnan, R., Madhyastha, H.V., Srinivasan, S., Jain, S., Krishnamurthy, A., Anderson, T., Gao, J.: Moving Beyond End-to-End Path Information to Optimize CDN Performance. In: ACM IMC (2009)
10. Allman, M., Paxson, V., Stevens, W.: TCP Congestion Control. RFC 2581 (April 1999) (Updated by RFC 3390)
11. Allman, M.: Tcp byte counting refinements. SIGCOMM Comput. Commun. Rev. (1999)
12. Huang, Y., Mehrotra, S., Li, J.: A Hybrid FEC-ARQ Protocol for Low-Delay Lossless Sequential Data Streaming. In: ICME (2009)
13. Padmanabhan, V.N., Mogul, J.C.: Improving HTTP Latency. In: WWW Conference (1994)
14. Mogul, J.C.: The Case for Persistent-Connection HTTP. ACM CCR (1995)
15. Cohen, E., Krishnamurthy, B., Rexford, J.: Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters. ACM CCR (1998)