Overclocking the Yahoo! CDN for Faster Web Page Loads

Mohammad Al-Fares[1,2] Khaled Elmeleegy[2] Benjamin Reed[2] Igor Gashinsky[3]
malfares@cs.ucsd.edu {khaled, breed, igor}@yahoo-inc.com

[1]*Department of Computer Science and Engineering* [2]*Yahoo! Research* [3]*Yahoo! Inc.*
University of California, San Diego

Abstract

Fast-loading web pages are key for a positive user experience. Unfortunately, a large number of users suffer from page load times of many seconds, especially for pages with many embedded objects. Most of this time is spent fetching the page and its objects over the Internet.

This paper investigates the impact of optimizations that improve the delivery of content from edge servers at the Yahoo! Content Delivery Network (CDN) to the end users. To this end, we analyze packet traces of 12.3M TCP connections originating from users across the world and terminating at the Yahoo! CDN. Using these traces, we characterize key user-connection metrics at the network, transport, and the application layers. We observe high Round Trip Times (RTTs) and inflated number of round trips per page download (RTT multipliers). Due to inefficiencies in TCP's slow start and the HTTP protocol, we found several opportunities to reduce the RTT multiplier, e.g. increasing TCP's Initial Congestion Window (ICW), using TCP Appropriate Byte Counting (ABC), and using HTTP pipelining.

Using live workloads, we experimentally study the micro effects of these optimizations on network connectivity, e.g. packet loss rate. To evaluate the macro effects of these optimizations on the overall page load time, we use realistic synthetic workloads in a closed laboratory environment. We find that compounding HTTP pipelining with increasing the ICW size can lead to reduction in page load times by up to 80%. We also find that no one configuration fits all users, e.g. increasing the TCP ICW to a certain size may help some users while hurting others.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: measurement techniques, performance attributes; C.2.3 [**Computer-Communication Networks**]: Network Operations—*network monitoring, public networks*

General Terms

Measurement, Experimentation, Performance

IMC'11, November 2–4, 2011, Berlin, Germany.

Keywords

Content Delivery Networks, Traffic Analysis, Web Page Load Time, TCP tuning

1. INTRODUCTION

For web sites to maintain high user satisfaction, their web pages need to load quickly. However, web pages are becoming more content rich over the past few years. They include many embedded images, scripts, and style sheets. Consequently, page load times are getting higher. As we will see in Section 4.1.2, many users can experience tens of seconds of load time for popular web pages like Yahoo!'s front page.

Our measurements and previous research show that virtually all of this time is spent in the network stack downloading the web page and its embedded objects [14, 20, 25]. Two main factors contribute to the long download time. The first is the network RTT from the user to the web server. Second, a page download typically takes tens of round trips to download the web page data and all its embedded objects. The number of round trips involved is called RTT multiplier.

Long network RTTs can be due to a combination of long Internet routes, route misconfigurations, and long queuing delays at routers along the packets' path. To alleviate this problem, CDNs are deployed across the globe by companies like Akamai, Facebook, Microsoft, Google, and Yahoo! These CDNs bring content closer to users across the world, hence reducing their network RTTs. However, this study and previous work by Krishnan *et al.* [16] have shown that even with globally deployed CDNs, many users experience hundreds of milliseconds RTTs. To alleviate this high RTT problem, Krishnan *et al.* proposed a tool, *WhyHigh*, that attempts to identify routing misconfigurations leading to inflated network RTTs. This helps fix these misconfigurations and reduce round trip times.

Inflated RTT multipliers are mainly due to inefficiencies during TCP slow start and in the HTTP protocol. TCP slow start probes the available link capacity by exponentially growing the transfer rate per RTT until a packet is lost (or the slow-start threshold is reached). This probing is fairly conservative, as it starts from a modest ICW, with a default value of three in most networks. Hence, it wastes many network round trips before the full available network bandwidth is utilized. Similarly, HTTP is used inefficiently in practice as it requests a single object at a time wasting a network round trip per object. For a page with tens of small embedded objects, this is very wasteful.

Two key optimizations were proposed by the IETF and industry [7, 11]: First, TCP should start probing from a larger ICW size.

Dukkipati *et al.* [11] argue for using an ICW of 10 segments. Using traffic measurements from Google's users, they argue that this would reduce object load time with virtually no downside. Second, TCP should open up the congestion window size at a higher rate per round trip. TCP slow start increases the congestion window by one for every acknowledgment received. However, delayed acknowledgments, which are pervasively deployed in the Internet, make the receiver send an acknowledgment for every other packet received. This causes TCP congestion window to increase by a factor of 1.5 instead of 2 per network round trip during slow-start. To remedy this problem, ABC [3, 5], was introduced to increase the window based on the number of bytes acknowledged instead of the number of acknowledgments received.

HTTP pipelining [12] was introduced to optimize HTTP downloads reducing the number of round trips. It allows for sending HTTP requests for new objects, while responses from earlier requests have not yet been recieved. As seen in Figure 1, HTTP pipelining saves RTTs reducing overall web page load time. Unfortunately, HTTP pipelining is not available by default in major web browsers. For example, Internet Explorer, the dominant web browser, does not support it. [1] And while Firefox supports it, it is disabled by default.

This paper is concerned with the delivery of content from edge servers from the Yahoo! CDN to the users. To this end, we collect packet traces of 12.3 million TCP connections from users of Yahoo! across the world. Using these traces, we present an in-depth cross-layer study of different factors affecting web page load times. Then, we study different cross-layer optimizations and their interplay aimed at reducing the RTT multiplier. Specifically, we study varying the ICW size, TCP ABC, and HTTP pipelining using live and realistically-inspired synthetic workloads.

The contributions of this paper are three fold:

1. Characterize the connections from users' to the Yahoo! CDN web servers at the IP, TCP, and HTTP layers.

2. Study TCP optimizations to reduce web page load times – most notably changing the ICW. We find that many users benefit significantly (up to 38%) from increasing the ICW size. However, in contrast to previous work, we show that no one size for the ICW fits all the users as increasing the ICW for some users can increase packet loss hurting the overall page load time. Moreover, we show that, in some cases, increasing the ICW size can be unfair to other flows in the network. We believe that currently this result is especially important given the efforts at IETF to increase TCP's ICW size to the fixed size of 10 [10].

3. Study and quantify the performance gains from HTTP pipelining using realistic workloads. In addition, quantify the gains when HTTP pipelining is used in conjunction with optimum ICW size. These gains can reach 80% reduction in the page load time.

The rest of this paper is organized as follows. Section 2 presents the background and previous related work. Section 3 characterizes the traffic observed at the Yahoo! CDN. Section 4 presents our study of different optimizations to reduce the RTT multiplier to reduce the web page load time. Section 5 discusses our findings. Section 6 concludes the paper.

[1]The main reason Microsoft gives is that pipelining is not universally implemented, e.g. head-of-line blocking with buggy proxy servers.

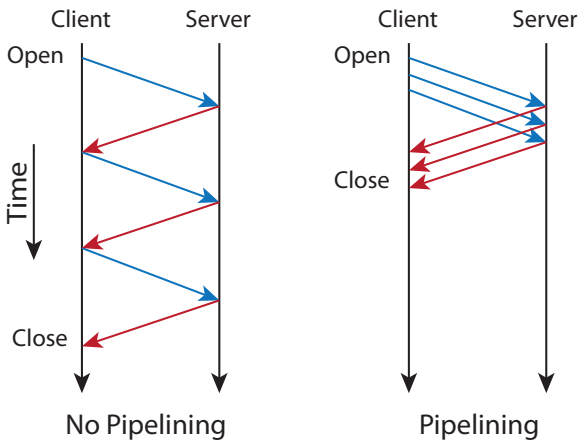

Figure 1: Non pipelined vs. pipelined HTTP connection. The client arrows indicate GET requests.

2. BACKGROUND AND RELATED WORK

In this section, we explain how content delivery networks work. We also present some of the related work aimed at optimizing content delivery at CDNs, more specifically optimizations to the network stack.

2.1 Content Delivery Networks

CDNs are usually built as a network of geographically diverse sites. Each site hosts a cluster of servers caching content and delivering it to users. The geographical diversity serves two purposes. First, it brings content closer to the users reducing network latency. Second, it provides redundancy to tolerate failures of individual sites.

In a nutshell, a CDN typically works as shown in Figure 2. When the user tries to fetch an object from a particular URL, it first performs a DNS lookup. The DNS server returns the IP address of a server near the user. The user then contacts the server to fetch the object. If the server has the object locally, it serves it to the user from its cache. Otherwise, the server contacts a back-end server, usually over a fast private network, to fetch the object into its cache and then serve it to the user.

There are multiple CDNs deployed worldwide. Some companies run and use their own private CDNs like Google, Facebook, and Yahoo!. Others use third party CDNs like Akamai [19] and CoralCDN [13].

In this paper, we study the delivery aspect of the Yahoo! CDN. At a high level, the Yahoo! CDN operates as described above.

2.2 Round Trip Times

Krishnan *et al.* [16] studied the network round trip latencies in the Google CDN. They reported that latencies are generally high and that 40% have round trip times higher than 400ms. They argued that adding more CDN sites is not always the best solution as this high latency is sometimes due to queuing delays and routing misconfigurations. They then introduced a new tool, *WhyHigh*, that tries to identify prefixes suffering from inflated latencies. Finally, this tool attempts to diagnose the causes for this inflated latency by using multiple active measurements, using different tools like *ping* and *traceroute*, and correlating inflated subnet latencies to common AS paths for example.

In this paper, we also studied round trip latencies, and also found latencies to be high (on the order of a few hundred milliseconds in the developing world). However, the latency distributions we observed were significantly lower than those reported in [16].

2.3 Optimizing the Network Stack

Previous work have argued for increasing TCP's initial window size [7, 10, 11]. Dukkipati *et al.* [11] recently argued for increasing the window size to 10 segments in order to decrease page load time. They argued that this reduces page load time with virtually no downside. In contrast, although we find that many users benefit from larger initial window size in this study, we also observe a subset of users who suffer due to increased packet loss.

Qian*et al.* [23], have studied the Internet backbone traffic and shown that up to 15% of large-flows already violate the ICW limit set by the spec ($min(4 * MSS, max(2 * MSS, 4380)$, which equals 3 for a Maximum Segment Size (MSS) of 1460) [7]), and values up to 9KB have been observed in the wild.

Allman [4] studied traffic to and from a single web server. He characterized different settings of the protocols used (TCP and HTTP) by this traffic. For example, he studied the deployment of TCP features like selective acknowledgments. Like other studies, Allman too reported long RTTs for studied connections. This study is over 10 years old though and only studied 751K connections at a single web server at a single geographic location. Moreover, unlike this paper, Allman only relied on passive measurements and did not try to measure different performance metrics in response to changing different protocol settings.

To allow for increasing the ICW while not hurting users with poor connectivity, Chu [10] *et al.* argued that users with poor connectivity can can advertise a smaller receive window sizes. In theory this can fix the problem. However, in practice, modifying the network stacks of existing users with poor connectivity to dynamically detect their network conditions and consequently adjusting their corresponding receive window sizes is challenging.

A different line of research proposed multiplexing several small streams on top of TCP to reduce web page load time, e.g. SPDY [2] and Stream Control Transmission Protocol [24]. However, both protocols are still experimental and not used at a large scale in the web. In contrast, we aim to optimize existing protocols to achieve best possible performance without breaking backward compatibility.

On the HTTP and application layer front, Leighton [17] advocates several optimizations such as pre-fetching embedded content, pre-caching popular objects at the edge, and using compression and delta-encoding of popular web pages. The argument being that, in contrast to a decade ago when the last-mile connection to the user was likely the bottleneck, the *middle-mile*'s capacity growth has not kept pace and become the new bottleneck, and that these techniques would all contribute to alleviating web traffic on the backbone and faster page loads.

3. STUDYING YAHOO! CDN TRAFFIC

In this section, we present our study of the traffic characteristics at the Yahoo! CDN edge servers. Specifically, we wanted to analyze and dissect network connections on multiple levels to answer questions as follows:

- Routing Layer (IP): What is the distribution of RTTs? Are some ISPs suffering from exceptionally high RTT to the nearest CDN node?

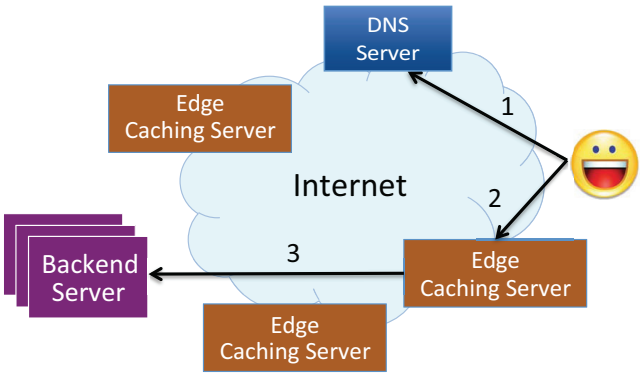

Figure 2: An HTTP request to a CDN. First, the DNS resolves the server's name to a nearby edge server. Then, the client sends the request to the nearby caching edge server. On a cache miss, the edge server contacts the back end servers to fetch the missing content and then deliver it to the user.

- Transport Layer (TCP): What level of packet retransmission rates do different users experience? What is the distribution of bytes transfered per connection? What is the distribution of the connection lifetime?

- Application Layer (HTTP): How many web objects are fetched per TCP connection? What is the distribution of the sizes of objects fetched?

3.1 Methodology

For this study, we used 1-hour long *tcpdump* traces collected from edge servers in the Yahoo! CDN across the world. We selected an edge server at each of the following sites: Chicago, Germany, Singapore, and India. These sites were chosen to span different important regions of the world with diverse connection characteristics. We have verified that the traffic characteristics at one server are representative of its corresponding site. We did this by collecting traces from different servers at the same site and comparing their characteristics and verifying they are virtually the same. Consequently, we only report results from one server per site. These servers run Linux with 2.6 kernels. Moreover, these servers were configured with default kernel settings for the TCP stack. The packet traces were collected at 1 p.m. local time, which previous studies at Yahoo! have shown to be traditionally the peak load time on these servers. We have also verified that traffic at other times of the day has qualitatively similar characteristics.

We used a combination of of *tcpsplit* [6] and *tcptrace* [21] to analyze every TCP connection we captured (12.3M connections). This provided a detailed report on a multitude of key connection characteristics; connection duration, number of bytes transferred, average roundtrip time estimates, retransmitted packets, etc. In addition, we used the HTTP module in *tcptrace* to parse HTTP layer information such as request arrival, response initiation, and completion timestamps, objects requests and their sizes, etc. Finally, we used Yahoo!'s proprietary internal data sets for geo-locations, connection speeds, and subnet prefixes in conjunction with information extracted from the traces to complete this study.

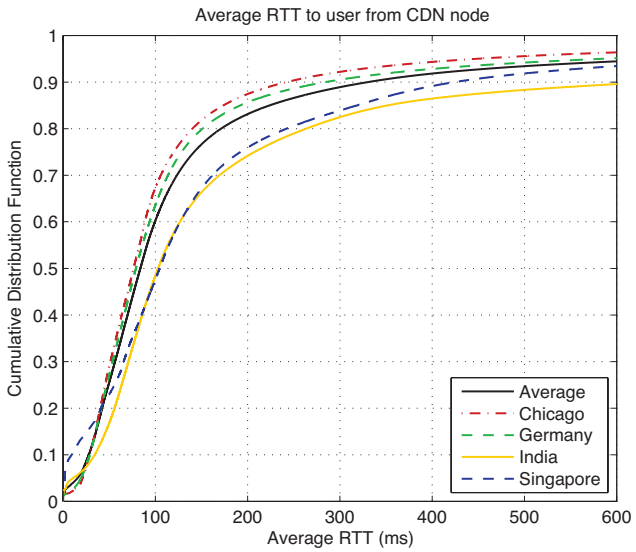

Figure 3: Average RTT distribution across the four Yahoo! CDN nodes.

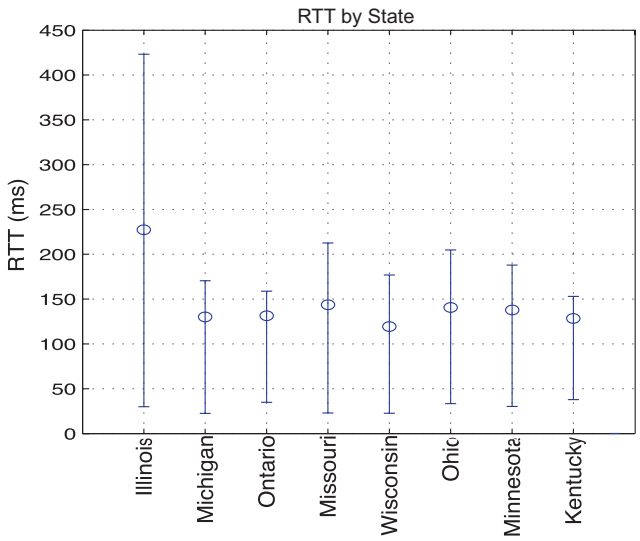

Figure 6: RTTs of top 8 most-frequent user origin states. Showing the median, 10th and 90th percentiles.

3.2 Results

3.2.1 RTT Characterization

In this section, we study the RTTs experienced by different users at the four sites.

Figure 3 shows the RTT distributions for the 4 sites. We note that the distributions for the Chicago and the Germany sites are considerably better than the Singapore and the India sites. As reported by previous work [16], we note that RTTs are generally high even though CDN nodes are geographically distributed to be close to the users. However, in contrast to [16], which showed that 40% of the user connections had greater than 400ms RTT, in our workload, only 10% of the user connections experience 400ms or more RTT.

Figures 4 and 5 breakdown the data by the users' source network (this applied clustering is similar to studies such as [15], which grouped web-clients based on source network, among other factors). They show the median, 10th, and 90th percentiles of RTT and packet retransmission rates of users' connections per source subnet for their corresponding sites. They also show the connection counts per subnet. Both figures only show the top-100 subnets with respect to the number of connections arriving from each subnet. Since both the Germany and the Chicago sites had similar connectivity characteristics, we chose to show one example of them only – Chicago. Similarly, we chose the Singapore site as an example of the other two sites. In both figures, we note that there is a wide range of RTTs with some subnets having connections experiencing multi-second RTT. Also, we notice a wide range of packet retransmission rates with some subnets having connections experiencing over 50% packet retransmission rate.

Figure 6 shows the RTT distribution of the 8-most frequent states connecting to the Chicago node. We note that even though these states are very close geographically to Chicago, big fraction of their connections experience hundreds of milliseconds RTT (This could be due to many reasons including long queueing delays). Hence, one can conclude that adding more CDN sites with geographical proximity to the users does not guarantee to significantly reduce their RTTs. This is consistent with observations and conclusions made by previous work [16].

Figure 7 shows the RTT distribution by connection type. Note that *Broadband* represents connections having high speed, yet their connection type is unknown. Also, note that mobile connections have significantly high RTT. Given the growth of mobile networks, improving the RTT multiplier for these connections becomes more pressing so that mobile users can have acceptable web page load times.

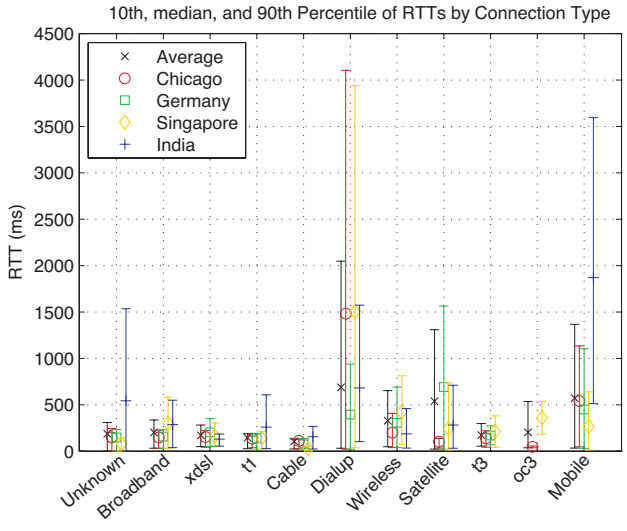

Figure 7: Median, 10th and 90th percentiles of RTTs, by connection speed.

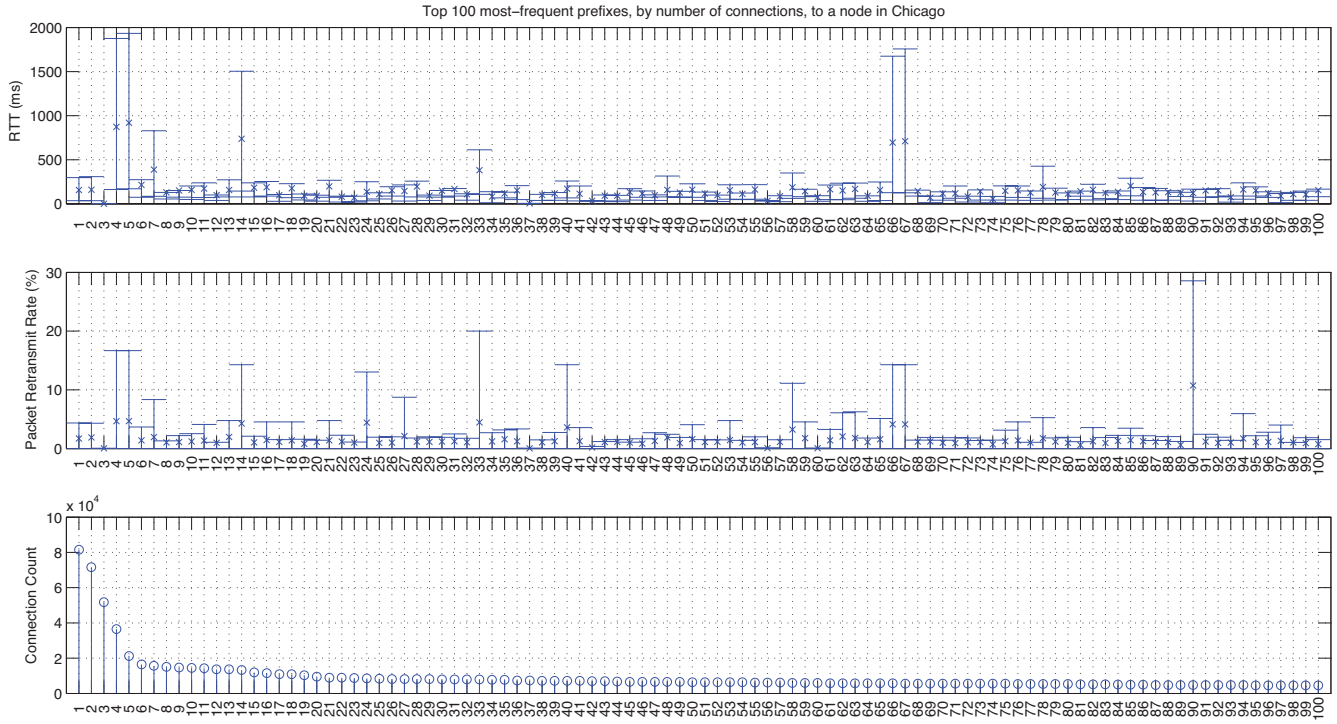

Figure 4: Median, 10th and 90th percentile RTTs, by user prefix, retransmission rate, and connection count to Yahoo! CDN node in Chicago.

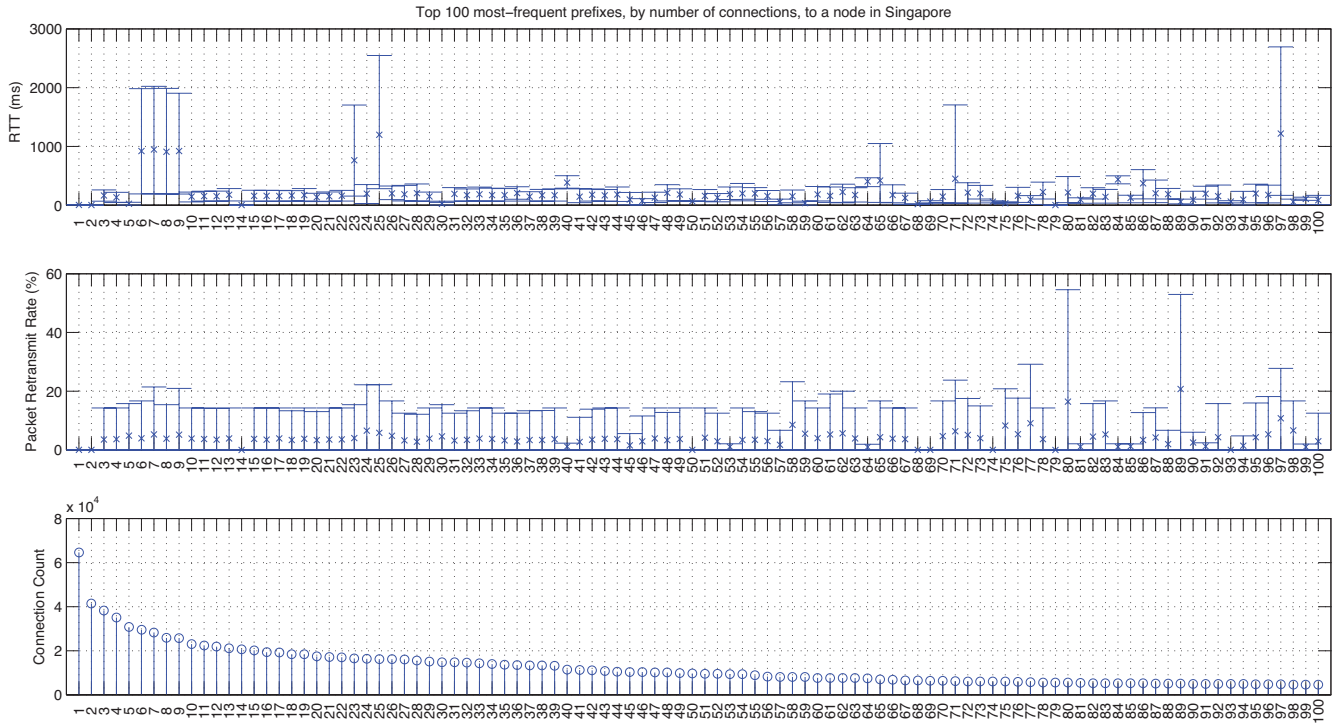

Figure 5: Median, 10th and 90th percentile RTTs, by user prefix, retransmission rate, and connection count to Yahoo! CDN node in Singapore.

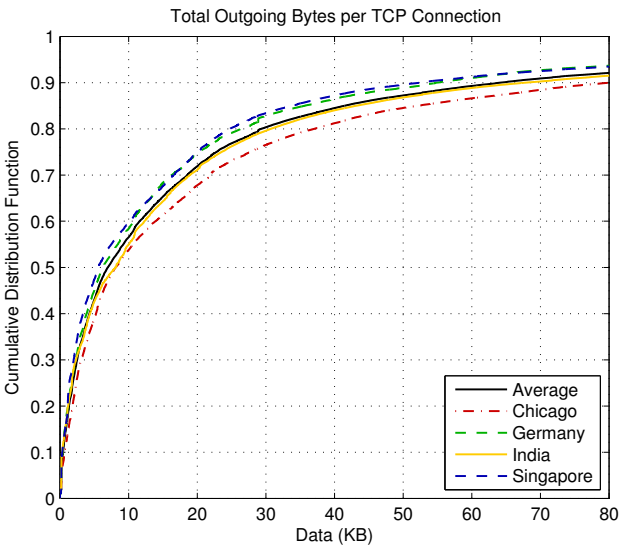

Figure 8: Bytes transferred per TCP connection.

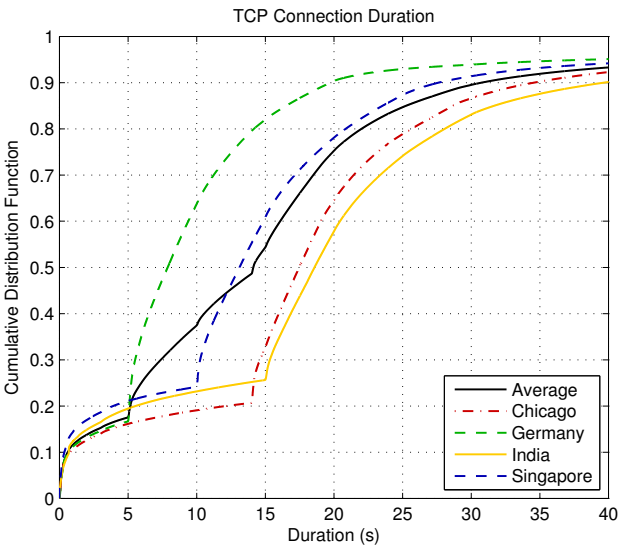

Figure 9: TCP connection duration.

3.2.2 TCP Connection Characterization

In this section we study different characteristics of users' TCP connections to the Yahoo! CDN.

Figure 8 shows the distribution of the total number of bytes transferred to the user per connection. We see that about 90% of connections download less than 64KB of data. Note that TCP Reno, the default TCP flavor at many operating systems including Linux, has 64KB as the default value for TCP's initial *ssthresh* (Slow Start Threshold). One important consequence of this is that 90% of the connections, barring loss, would never enter TCP's congestion avoidance phase and the transfer is done entirely in the slow-start phase.

Figure 9 shows the distribution of connection duration at the four servers. The different knees in the graphs correspond to the server's connection timeout setting, which reflects differences in the local server configurations. Note that this does not correspond to active transfer times; due to the typical size of objects requested, and the small number of objects in a persistent connection, as we will see in Section 3.2.3, a typical connection is idle most of the time.

To study packet loss for these connections, we use a metric that is measurable from the packet traces we collected, i.e. retransmission rate. This retransmission rate is an upper bound on the packet loss rate. Since most users use selective acknowledgments, retransmissions establish a tight upper bound. Figure 10 shows the distribution of packet retransmission rates per connection. Note that, in India for example, over 70% of the connections see no retransmissions; however, over 17% of connections have retransmit rates above 10%. Similarly, in Figures 4 and 5, we see that some subnets experience very little retransmissions, while others experience substantial retransmission rates that sometimes reach 50%. As we see in Section 4.1.2, overall page load time is extremely sensitive to the packet-loss rate, especially during connection setup, where a SYN timeout is on the order of seconds. This is compounded for networks where RTTs are significantly higher, and it is not uncommon to see total page load time in the range of 10-120 seconds for the Yahoo! frontpage.

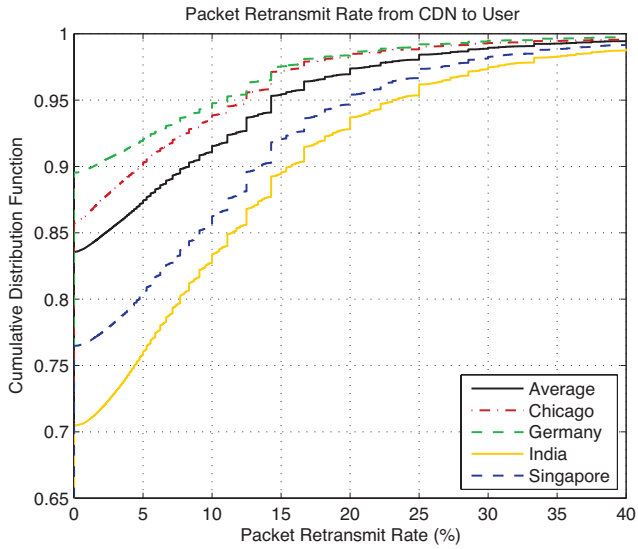

Figure 10: Packet retransmission rate.

3.2.3 HTTP Workload Characterization

In this section we study different properties of the downloaded web objects at the Yahoo! CDN. We can see the distribution of requested object sizes in Figure 11, which shows around 90% of objects are smaller than 25KB (17 segments). Figure 12 shows the distribution of the number of HTTP requests per connection to see the effect of persistent connections. The mean is only about 2.4 requests per connection, with the majority of connections requesting only one object. The reason for having a small number of requests per connection in spite of typical web pages having tens of objects is because web browsers typically use multiple concurrent TCP connections per domain per web page. Putting together Figures 11 and 12 tells us that even when requesting multiple objects back-to-back, objects are so small and so few that a typical connec-

tion does not have enough time to take advantage of opening up the congestion window. Hence, most of the time is spent ramping up in slow-start.

In Figure 13, we show the distribution of the time between HTTP requests within the same connection (so-called "think-time"). We observed that the overall majority, about 80% of back-to-back requests, occur in under one second, and therefore unlikely to be the result of user-clicks, but rather the browser fetching objects.

Linux 2.6 kernels also provide a setting called *tcp_slow_start_after_idle*, which resets the congestion window to the ICW and moves TCP back to the slow-start phase if there is no data to send after a given idle period, defined as one retransmission timeout (RTO). This is *on* by default. In Figure 13, we also plot the distribution of the difference between the inter-request time and our estimate of of the RTO, calculated using the standard Jacobson estimator: $RTO = RTT_{average} + 4 * RTT_{variance}$. We find that approximately 10% of back-to-back object requests are seperated by more than one RTO. All these users have to go through slow start again when downloading the following object spending expensive network round trips to probe the network for bandwidth again.

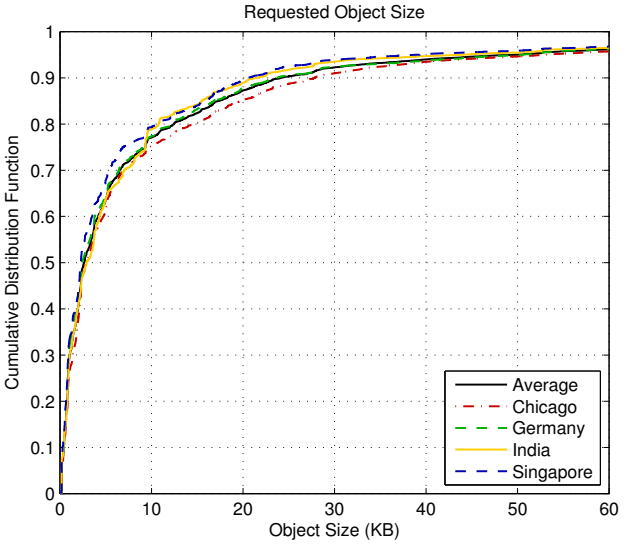

Figure 11: HTTP object size distribution.

Figure 14 shows the distribution of the fraction of the total number of bytes downloaded via objects with certain sizes. For example, we note that 50% of the total bytes downloaded come from objects with sizes of 60KB or greater. Looking at Figure 11, we note that less than 5% of the web objects downloaded have sizes of 60 KB or more. Hence, one can conclude that less than 5% of the web objects downloaded account for 50% of the bytes downloaded.

4. IMPACT OF DIFFERENT OPTIMIZATIONS ON WEB PAGE LOAD TIME

In Section 3, we saw that RTTs are generally high. They are even higher for the up-and-coming segment of users – mobile users. For CDN operators, little can be done about this to significantly change the picture. This means that the bigger opportunity to reduce the web page load time lies in reducing the RTT multiplier. In this section, we study different optimizations to reduce this multiplier.

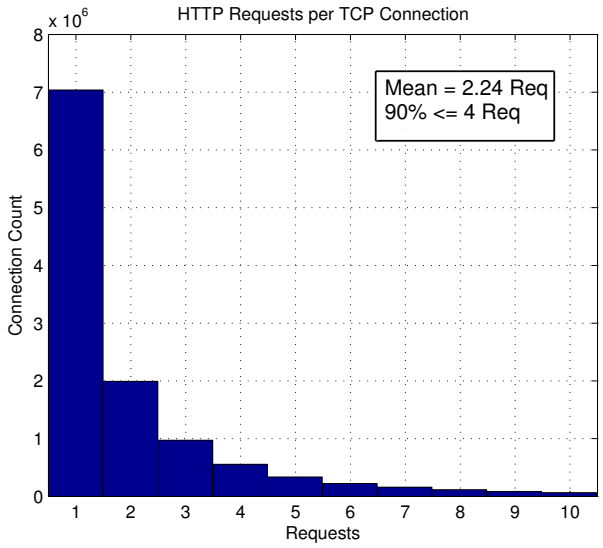

Figure 12: HTTP requests per connection.

Two of these optimizations attempt to reduce the number of round trips during TCP slow start.

The first optimization is increasing TCP ICW size, which attempts to make TCP slow start begin transmitting data at a rate closer to the maximum available bandwidth. Hence, it can use fewer round trips to reach optimal window size that achieves the maximum transmission rate. Section 4.1 studies the effects of using larger ICW sizes.

The second optimization is opening up the congestion window at a higher rate during slow start. TCP slow start increases the congestion window by one for every acknowledgment received. However, delayed acknowledgments, which is pervasively deployed in the Internet, makes the receiver send an acknowledgment for every other packet received. This causes TCP congestion window to increase by a factor of 1.5 instead of 2 (as originally intended) per round trip. To remedy this problem, Appropriate Bytes Counting (ABC) was introduced. ABC increases the window based on the number of bytes acknowledged instead of just counting the number of acknowledgments received. In Section 4.2, we study the effectiveness of ABC in reducing page load time.

Finally, in Section 4.3 we study the effectiveness of HTTP pipelining in reducing the RTT multiplier. Moreover, we study its interplay with increasing the ICW size.

For these different optimizations, we experimentally evaluated their effectiveness using live traffic from real users. Moreover, we used macro benchmarks to evaluate their overall effects on web page load times in a closed laboratory environment. In these macro benchmarks, we constructed our synthetic workloads based on measurements from the live traces.

4.1 Initial Congestion Window (ICW)

We have seen that the vast majority of Yahoo! CDN connections transfer very few, and very small objects, which means that TCP spends most of its time in the slow-start phase. For this reason, improving the efficiency of this phase is crucial. When a new TCP connection starts probing for available bandwidth, the Linux TCP implementation follows RFC 3390 [7], which specifies an ICW of 3 segments for networks having a MSS of 1460 bytes, which is the

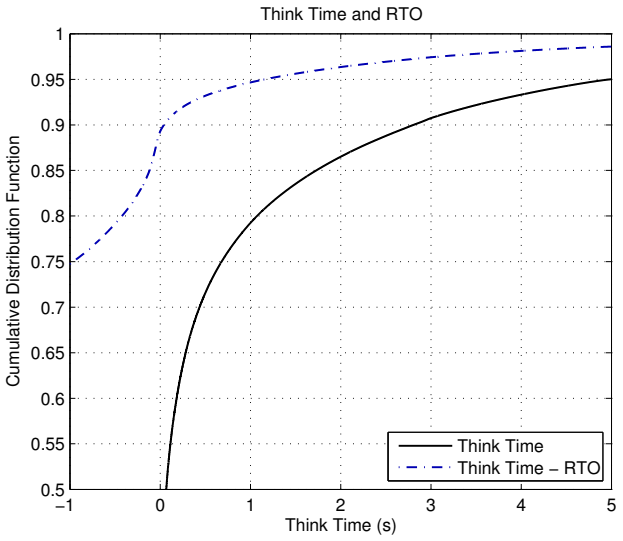

Figure 13: Inter-request time and retransmission timeout (RTO).

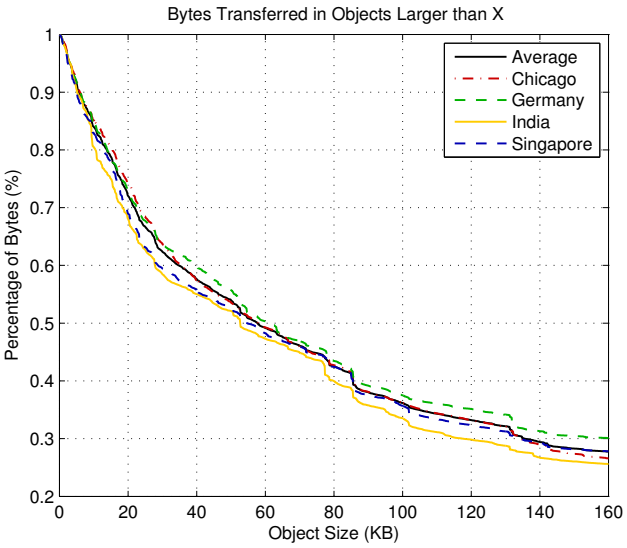

Figure 14: Percentage of overall bytes transferred across all objects vs. object size.

most common MSS. By increasing this ICW, small objects stand to be transferred in fewer RTTs, which when compounded across all objects on a page can cut down the total page load time significantly.

Obviously, TCP at the server end would not send more unacknowledged data than is allowed by the client's advertised receive window so as not to overflow the client's receive buffer (The receive window is dynamically allocated on most operating systems and advertised throughout the connection). Luckily, on popular operating systems (except Linux which has a much smaller receive window), the initial receive window is quite large (64KB-256KB), which would allow for utilizing a larger ICW. According to [11], more than 90% of user connections have receive windows large enough to utilize an ICW of 10. Hence, increasing TCP's ICW can be beneficial.

4.1.1 Evaluation Using Live Traffic

To test the effects of increasing the ICW size, we chose one of our CDN sites – Singapore. We chose it due to the diversity of its connections qualities as seen in Figure 5. There, we varied this setting and captured traces for each ICW size.

We show in Figure 15 the distribution of object transfer time normalized by the connection's average RTT. This normalization allows us to compare transfer times of objects over connections having different RTTs. We observe a reduction of 32% in object transfer time overall at the 80th percentile when going from an ICW of 3 to 16. However, increasing the ICW sees diminishing returns beyond that point. Because the effects of increasing the ICW would be more evident during the beginning of a TCP connection, we show in Figure 16 the same metric for the *first* HTTP request only, where the improvement is a 38% reduction in transfer time. Subsequent HTTP requests benefit less as the TCP window is usually opened up at the first request. Note that in both Figures 15 and 16 there is a fraction of the objects that are downloaded in under one average RTT. The reason for this is that the average RTT is measured across all round trips measured in the lifetime of a connection. By examining the traces, we observed that sometimes round trip times vary significantly within a single connection. Hence, within a single connection an RTT can be less than the connection's average RTT.

A potential side effect of increasing the ICW is an increase in retransmission rates, which we did observe as shown in Figure 17. Note that while over 70% of connections see no retransmissions, increasing the ICW from 3 to 16 increases the retransmit rate from 17% to 25% for about 10% of connections. As we will see later in this section, this will have a significant impact on the overall page load time as it is highly sensitive to packet-loss.

To study the effects of increasing the ICW size on the tail of the distribution of object transfer times on a per-subnet basis, we show in Figure 18 the 80th percentile of the average object transfer times per network prefix. In this figure, for each subnet, 7 data points, corresponding to 7 different ICW sizes, are presented. Each point represents the average object transfer time normalized by the average RTT. In the figure, we can observe that most subnets benefit from increasing the ICW and see dramatically lower object transfer times. We also note that for these 20% of the objects, maximum benefit is achieved at an ICW of 16. After that, for ICW of 32, object transfer time goes higher. In contrast, other objects are not hurt by larger ICW sizes as seen in Figure 15. Note that we chose the *80th* percentile to show that a significant portion of the connections in the tail of the distribution can suffer from using larger ICW size.

4.1.2 Studying Page Load time

While the previous section studied traces of individual HTTP requests in the wild, we also wanted to capture the effects of tuning the TCP ICW size on the overall page load time. Since a full web page encompasses many objects, any straggling object download will delay the overall web page download time, especially if this straggling download is for the HTML file. Studying whole page load times in the wild is very difficult though. This is because when using packet traces, there is no notion of a full page download as the objects in a page likely span multiple connections that are difficult to tie together in a postmortem analysis. Moreover, these objects are often at different servers. Hence, no one server

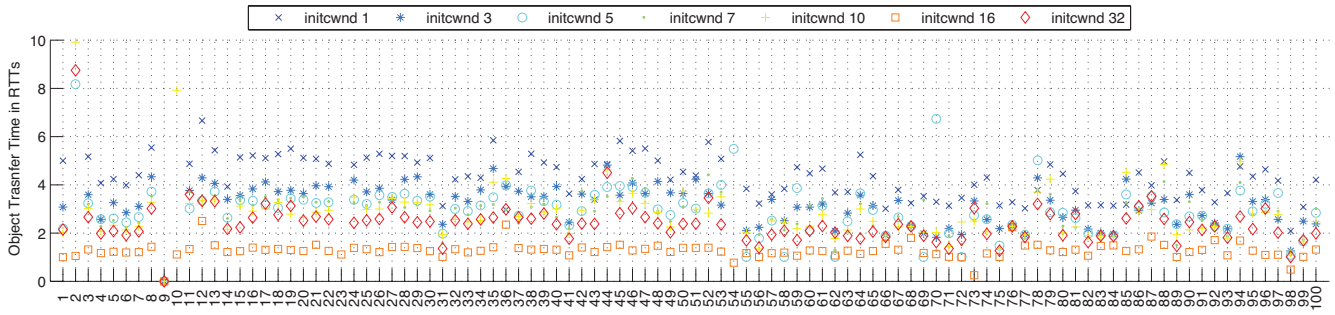

Figure 18: The 80th percentile of object transfer times in RTTs per network prefix for the top 100 prefixes, for different ICW sizes.

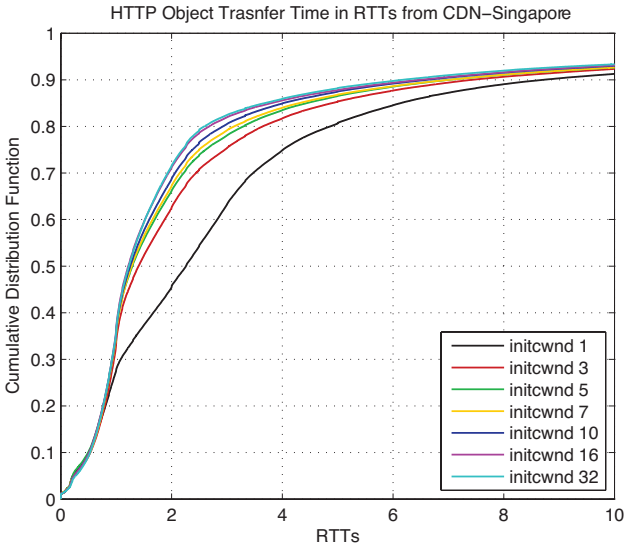

Figure 15: Overall object transfer time in RTTs.

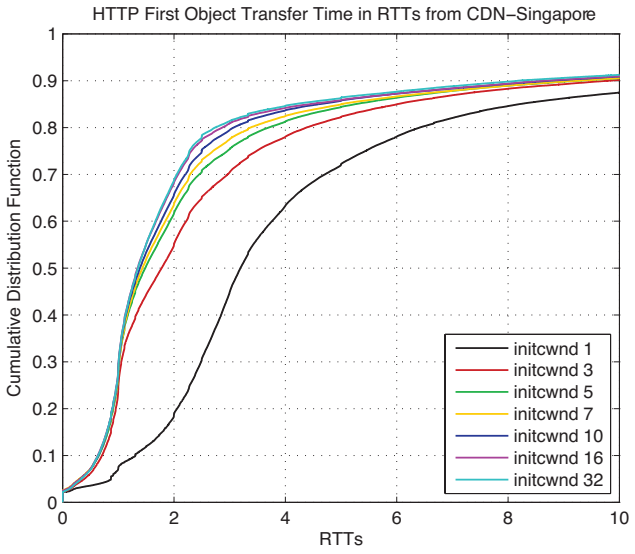

Figure 16: First-object transfer time in RTTs.

trace can contain full page downloads. For this reason, this section studies the overall page load time in a controlled environment in the lab using real workloads and realistic network conditions. These simulated network conditions (latency and packet loss rates) are based on connections' profiles from previous sections. These network conditions represent links' latecy and congestion due to cross traffic in the internet.

Experimental Setup

We captured a snapshot of the Yahoo! front page, by far the most popular Yahoo! web page, and fairly representative of the measured traffic workload. We hosted this page and its resources locally such that all object requests are directed to a local server.

In this setup, the client browser is 3.6.9 Firefox running on Mac OS X 10.5.8. Apart from disabling caching, all other browser and TCP settings are set to their default. Most notably, delayed ACKs are enabled, and for Firefox, six maximum simultaneous connections per domain were used.

The server is a guest VM on the same machine, running CentOS 4.8 release with an updated 2.6.29.5 linux kernel and the apache 2.0.52 web server. We used the *ipfw* command to setup dummynet pipes in order to control the perceived RTT and packet-loss rates from and to the server. During the experiments, no hardware resource was saturated, which guaranteed that request latencies were only due induced RTTs and packet loss. The web page had 30 objects distributed across two domains. Thus, we created two IP aliases at the server representing the two domains.

We had the client repeatedly request the front page. The embedded links , which represented the static content on the page, namely images, Javascript, and stylesheets, were replaced to point to IP aliases on the local server. Firefox fetched 30 objects distributed across two domains via six concurrent connections per domain resulted in an average of 2.5 objects per connection, very close to the average number of objects per connection measured in live traces shown in Figure 12.

We wanted to measure the total page load time, which we define as the time difference between the first packet of the first HTTP request and the final ACK of the last object fetched (Page rendering time is negligible compared to the download time). We captured the tcpdump trace of the request at the client, and reported the page load time between the first outgoing request and the last object ACK. For every parameter combination we reported, we gave the geometric mean of five requests for the Yahoo! front page.

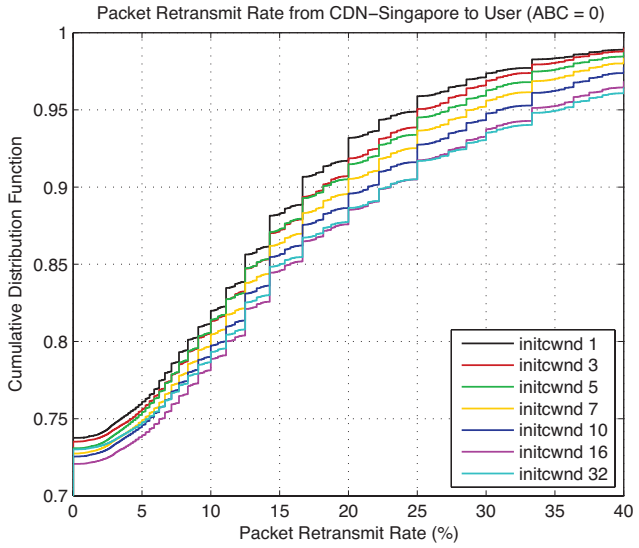

Figure 17: Packet retransmit rate.

Results

Figure 19 shows the total page load time for different values of RTT and different values of ICW sizes. We find that the relative reductions in page load times were relatively consistent; ranging from 27%-38% when going from an ICW of 3 to 16.

When taking packet loss into account, we show in Figure 20 the different page load times for different loss rates and different ICW sizes and an RTT of 100ms (the median for Singapore). We find that page load times and their variance appear to be superlinear and very sensitive to packet loss; increasing the loss rate from 5% to 10% *increases* page load time by 63% for an ICW of 3.

However, as seen in Figure 17, increasing the ICW can increase packet loss, especially for users with congested links. This raises important questions. First, can increasing the ICW hurt the overall page load time for some connections? Second, if the answer to the previous question is *yes*, is there a single optimum value for ICW with respect to page load time that fits all connections?

To study whether there are cases where increasing the ICW size can hurt the overall web page load time, we need to consider users that would suffer from increased packet loss due to increasing the ICW. For such users, we need to estimate the increase in packet loss due to the increase in ICW. Then, based on the estimated packet loss and the corresponding ICW, we should measure the corresponding page load time. To this end, we use a user connection in the 90% percentile of retransmission rate in Figure 17 as an example. We assume that the measured retransmission rate is equal to packet loss rate. Moreover, we assume that increasing the ICW size for this connection will follow the same increases in retransmission rates observed for the 90th percentile connection in Figure 17. The pairs of the ICW sizes and the corresponding packet loss rates are listed in the first two columns of Table 1. Finally, we assumed that this connection has the median RTT for the Singapore site – 100ms. We measured the page load times using the experiment described above for every ICW size and its corresponding packet loss rate listed at Table 1 and recorded it in the table at the third column. We note that for the 90th percentile connections (with respect to packet retransmission rates), increasing the ICW can lead to significant increase of page load time. The same would apply to other connections with higher percentile retransmission rates. We see that for the connection in the 90th percentile, it first benefits from increasing the ICW size up to seven, then by increasing the ICW size more, the page load time starts increasing until it reaches 70% more than the minimum load time achieved at an ICW size of 7.

Conversely, looking at the right side of Table 1 (columns 4 and 5) with zero packet loss rate (representing the 70th percentile retransmission rate of all connections) we see the benefits from increasing the ICW size all the way to 32.

Consequently, one can conclude that no one ICW choice would benefit all users.

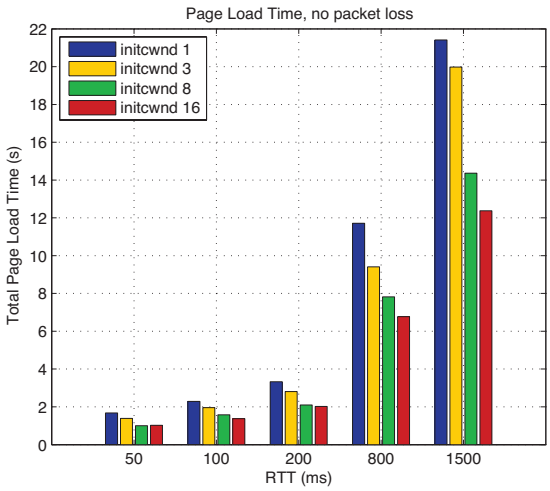

Figure 19: Page load time for different ICWs with no loss.

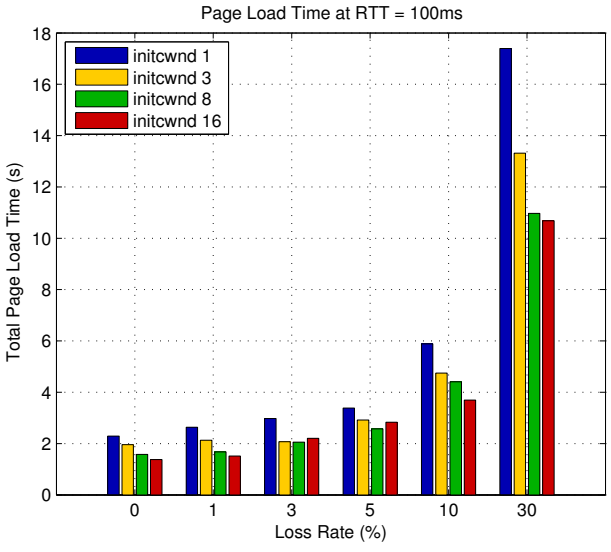

Figure 20: Page load time for different ICWs, packet loss rates, and an RTT of 100ms (median for Singapore).

initcwnd	loss (%)	Time (s)	loss (%)	Time (s)
1	17	8.56	0.00	2.36
3	18	8.06	0.00	1.87
5	18	7.29	0.00	1.72
7	20	6.32	0.00	1.56
10	22	6.92	0.00	1.50
16	25	10.22	0.00	1.32
32	25	10.87	0.00	0.96

Table 1: Yahoo! front page load times for an RTT of 100ms (median for Singapore) with increasing ICWs and their corresponding loss rates from Figure 17. Right columns show times with no loss for comparison.

4.1.3 Impact on TCP Fairness

When making changes to TCP, we need to make sure that it remains fair to other TCP flows in the network. In this section, we study the impact of increasing the ICW on TCP fairness.

Increasing TCP ICW size can be unfair to longer flows, sharing the same bottleneck with short flows. If the network is congested and experiencing significant packet loss, increasing the ICW will increase packet loss. This loss may cause the congestion window to shrink by half each round trip until it may eventually reach one. Moreover, the sender could even experience timeouts. Hence, for a long lived flow, the sender can end up sending at a rate lower than one packet per round trip. Conversely, for a short flow, e.g. a web object download, packet loss will only extend the transmission by a few round trips and the window size may not drop to one by then. Hence, the *average* window size for the whole transfer can be significantly higher than that for a large transfer.

To demonstrate this point, we conducted the following experiment using the setup described at Section 4.1.2. We configured the connection between the host machine and the virtual machine to have the profile of connections in the 95th percentile from Figure 10, i.e. 100ms of RTT and 25% packet loss rate. Also, we configured the ICW to be 32.

First, we ran *iperf* for 5 minutes between the two machines representing a long flow. This transfer achieved a bandwidth of 12.6KB/s – less than 1 segment per RTT (14.3KB/s). For the second experiment, we downloaded a 48KB (32 segments) file off the web server. The measured bandwidth for this second transfer was 59.1KB/s. Note that as discussed in Section 3.2.3, although 48KB is the 95th percentile of downloaded web object sizes, it is the 50th percentile with respect to objects contributing to overall bytes downloaded from the Yahoo! CDN web servers.

Moreover, given that a recent study has shown that 52% of the internet traffic is web traffic [8] and given that increasing the ICW can increase packet loss and congestion for users with poor connections as shown in Fig. 17, we conclude that increasing the ICW can be more unfair for longer flows for users having poor connectivity.

Furthermore, one can conclude that increasing the ICW will be unfair to other short flows that remain using small ICW – e.g. the current default value of 3.

We conclude that if the TCP standard is changed advocating larger ICW sizes, this will be unfair to some flows, e.g. long flows having high packet loss.

4.1.4 Discussion

In this section, we have seen that varying ICW size can have a significant impact on web page load times. While a big fraction of users benefit from larger ICW sizes, others may suffer. Moreover, the level of benefit or suffering can vary across users according to their connection quality. Consequently, to achieve best performance, different users should have different values of ICW depending on their connection quality. Hence, we argue that a dynamic mechanism for configuring the ICW per connection is needed.

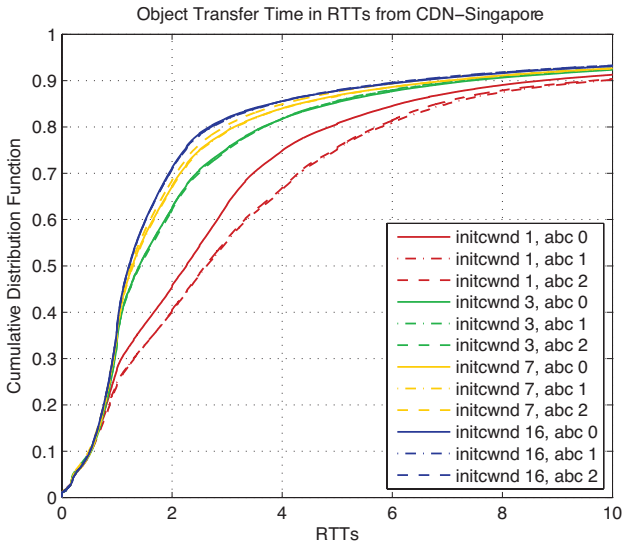

Figure 21: First object transfer time in RTTs, for different ICW and ABC settings.

4.2 Appropriate Byte Counting (ABC)

As explained above, the deployment of *delayed acknowledgments* is pervasive. This leads to the congestion window growing only by a factor of 1.5 per round trip. ABC mitigates this effect by relying not on ACK arrival but on the number of bytes acknowledged instead to increase the window size. This results in the congestion window doubling during slow-start every round trip as it was intended in the original TCP design. Linux allows 3 ABC settings: 0 turns off ABC; 1 increments the window by one for each acknowledged full segment; 2 increments the window by two for each acknowledgement received.

4.2.1 Evaluation Using Live Traffic

We varied the ABC settings at the server and show their effect in Figure 21. From these results we find that turning this feature on has a positive, but limited, effect on object transfer times.

4.2.2 Studying Page Load time

To study the effects of TCP ABC on overall web page load time, we used the same setup used in 4.1.2, except we used an RTT of 100ms and packet loss of zero and measured the page load time with the ABC optimization turned on and off. As shown in Figure 22, turning on the ABC optimization has marginal effects on the overall web page load time. The reason for this is that the effects of ABC will only be noticeable after many round trips. However, each TCP connection downloads very small number of little objects, and thus it requires only few packets and few round trips. Hence, there is not much difference in transmission time.

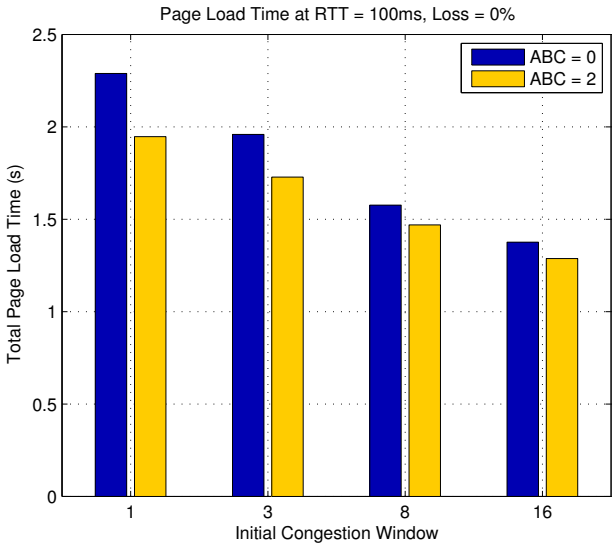

Figure 22: Page load time, for different ICW and ABC settings.

4.3 HTTP Pipelining

HTTP pipelining is known to deliver better performance. In this section, we quantify the performance gains from using HTTP pipelining using realistic workloads. We also study the interplay of HTTP pipelining with lower layers optimizations like increasing the ICW size.

HTTP 1.0 only supported downloading a single object per connection. This was inefficient especially for high delay networks as at least two network round trips are spent per object download – one for TCP connection establishment and another for downloading the object. Moreover, web pages are typically composed of multiple small objects, making HTTP 1.0 even less efficient for latency purposes. Later, HTTP 1.1 supported persistent connections spanning multiple object downloads. Furthermore, it allowed for pipelined HTTP requests over the same TCP connection as shown in Figure 1.

HTTP pipelining can eliminate multiple expensive network round trips from a web page download. This is because the client can request multiple objects without waiting for the corresponding responses to arrive from the server as seen in Figure 1.

Furthermore, pipelining allows web downloads to utilize much larger TCP ICW sizes leading to faster downloads. As seen in Section 4.1, currently the vast majority of web downloads cannot utilize TCP ICW larger than 16, even if the web user has more network bandwidth available. This is because web browsers download one object at a time. As seen in Figure 11, 90% of objects downloaded from the Yahoo! CDN are less than 24KB, i.e. they can fit in a TCP window of 16. In contrast, with pipelining, multiple objects with much larger aggregate sizes are downloaded concurrently. Hence, much larger windows can be utilized to reduce the number of network round trips per download.

Unfortunately, most web browsers do not support HTTP pipelining. For example Internet Explorer, the most popular browser [1], does not support it. Firefox, the second most popular browser, claims to support pipelining but has it switched off by default. As of today, these two browsers control more than 74% of the web browser market [1]. We tried to verify the level of pipelining supported by Firefox. We turned pipelining on, configured Firefox to use a single connection to download objects from each domain, and then downloaded Yahoo!'s front page. By looking at packet traces of the download, we realized that only a few object downloads were pipelined – at most two at time. This is in spite the fact that the web page had many objects per domain – up to 15 objects.

A main reason for not supporting pipelining by web browsers is that some web proxies in the Internet do not support it. However, this problem can be overcome if web browsers are modified to probe a well known web server, at start time, to check if it is connected to the server via a faulty proxy or not [18]. If no faulty proxy is found, pipelining is used, otherwise, it is not.

To study the effectiveness of HTTP pipelining and its interplay with TCP ICW size, we evaluated it experimentally. We built a simple web client that implements HTTP pipelining. It first downloads the HTML files. Then, it connects to the domains hosting the different web page objects. All objects per domain are downloaded in a pipelined fashion over a single persistent TCP connection. We used the setup in Section 4.1.2 having Yahoo!'s front page as our workload. The page objects span multiple domains. The domain with the maximum aggregate objects' size had 14 objects with total aggregate size of 445KB. To allow for a single network round trip download, we set the TCP ICW size at the server to 300 ($\sim$450KB). We also set the web client's receive buffer to 450KB. Under these settings, the web page download took 4 round trip times. The first two round trips were to connect to the server then download the HTML file. Once the HTML file is downloaded, its embedded objects and their locations are identified. The client then starts parallel connections to all the domains involved, one connection per domain. For each domain, HTTP pipelining is used to fetch all the objects in question at this domain. Hence, The two last round trips are for connecting to the domain, then fetching all the objects. This is well below the minimum number of round trips obtained in Section 4.1 – 8 round trips.

It is worth mentioning that this minimal number of network round trips can be also achieved by downloading all the web page objects concurrently each via a separate TCP connection. Current web browsers are moving in this direction by having multiple TCP connections per domain. This is why we see in Figure 12 that the average number of object downloads per TCP connection is 2.4. However, this approach has many shortcomings. First, it limits TCP's ability to control congestion as TCP controls congestion within single connection. So, if many connections start transmitting packets concurrently congestion collapse can happen, especially if the connections have high ICW. In this case a separate congestion manager [9] may be needed to control congestion across all the connections, which increases the system's complexity. Second, having multiple concurrent connections consumes more resources, e.g. the per connection TCP state and the CPU cycles to maintain this state. That is why most web browsers cap the number of concurrent connections used for objects download. For example, by default, Firefox caps the number of persistent connections per domain to 6 and caps the total number of connections to 30. Web browsers for mobile phones use less concurrent connections, e.g. Safari for the iPhone uses 4 connections. While the load from 6 connections per domain may be not that significant for most clients, it is certainly significant on web servers. It effectively means that they have to handle 6 times more connections and their corresponding overhead. Finally, using multiple TCP connections per application can be unfair to other applications and users that use a single connection per application.

5. DISCUSSION AND FUTURE WORK

As we have seen in the previous sections tuning the initial congestion window size can have a great impact on web page load time, especially if used in conjunction with HTTP pipelining. In Section 4.3, we have seen that web page load time can benefit from huge ICW sizes measured in hundreds of packets. Whereas, we see in Figures 18 and 17 that some users suffer from having much smaller ICW sizes.

This wide range of optimal initial TCP window sizes calls for a dynamic scheme for setting this size per connection as there is no one size that fits all. The right size depends on the connection characteristics, e.g. available bandwidth. This information can be learned from history of connections coming from the same location. TCP Fast Start [22] was proposed to allow a new connection to a host to reuse the state of older connections to the same host. It also included a mechanism to modify intermediate routers to identify packets from hosts using Fast Start and give these packets lower priority in the event of congestion. This is to account for the fact that Fast Start may be using stale cached information. It is questionable though how effective this reuse will be. A typical host does not reconnect to the same other host very frequently. Hence, *per-host* cached information is likely to be stale. Moreover, maintaining persistent state per client may be very expensive for a server serving millions of clients. Finally, the requirement for modifying intermediate routers is a significant hurdle to adoption.

We believe that the right approach to setting the initial TCP congestion should rely on previous history as well. However, a more effective approach tailored for web servers is needed. We leave this as future work.

Finally, because of its effectiveness, we advocate supporting pipelining in web browsers and implementing techniques, like [18], to overcome faulty proxies. Moreover, it will allow for taking full advantage of larger ICW sizes.

6. CONCLUSION

In this paper, we first characterized the traffic workload observed at the edges of Yahoo!'s content distribution network. We noticed that many connections have high RTTs. Some had significant retransmission rates. Based on this, we suggested and evaluated, both in the wild and in the lab, the effects of several optimizations at the TCP and HTTP layers with the goal of reducing overall page-load time. These included a combination of TCP Appropriate Byte Counting, increasing the slow-start ICW, and HTTP pipelining.

Overall, we find that, based on our traffic studies, a majority of users would see significant benefit from increasing the ICW – up to 38% reduction in page load time. However, for clients in poorly connected networks with high packet loss-rates, performance is likely to suffer when using high ICW sizes. For this reason, we conclude that no "optimal" setting exists that satisfies all users. We also found that HTTP pipelining is very effective, especially if used in conjunction with large ICW sizes. This combination can reduce page load time by up to 80%.

7. REFERENCES

[1] Global web stats. http://www.w3counter.com/globalstats.php, 2010.

[2] SPDY: An Experimental Protocol for a Faster Web. http://dev.chromium.org/spdy/spdy-whitepaper, 2010.

[3] ALLMAN, M. Tcp byte counting refinements. *SIGCOMM Comput. Commun. Rev. 29* (July 1999), 14–22.

[4] ALLMAN, M. A web server's view of the transport layer. *SIGCOMM Comput. Commun. Rev. 30* (October 2000).

[5] ALLMAN, M. TCP Congestion Control with Appropriate Byte Counting (ABC). RFC 3465, IETF, 2003.

[6] ALLMAN, M. tcpsplit. http://www.icir.org/mallman/software/tcpsplit/, 2010.

[7] ALLMAN, M., FLOYD, S., AND PARTRIDGE, C. Increasing TCP's Initial Window. RFC 3390, IETF, 2002.

[8] ARBOR NETWORKS. 2009 Internet Observatory Report. http://www.nanog.org/meetings/nanog47/presentations/Monday/Labovitz_ObserveReport_N47_Mon.pdf, 2010.

[9] BALAKRISHNAN, H., RAHUL, H. S., AND SESHAN, S. An integrated congestion management architecture for internet hosts. In *Proceedings of SIGCOMM '99* (New York, NY, USA, 1999), ACM.

[10] CHU, J., DUKKIPATI, N., CHENG, Y., AND MATHIS, M. Increasing TCP's Initial Window. http://tools.ietf.org/html/draft-ietf-tcpm-initcwnd-01, 2011.

[11] DUKKIPATI, N., REFICE, T., CHENG, Y., CHU, J., HERBERT, T., AGARWAL, A., JAIN, A., AND SUTIN, N. An argument for increasing tcp's initial congestion window. *SIGCOMM Comput. Commun. Rev. 40* (June 2010).

[12] FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T. Hypertext Transfer Protocol – HTTP/1.1, 1999.

[13] FREEDMAN, M. J., FREUDENTHAL, E., AND MAZIÈRES, D. Democratizing content publication with coral. In *Proceedings of NSDI '04* (Berkeley, CA, USA, 2004), USENIX Association.

[14] HOPKINS, A. Optimizing Page Load Time. http://www.die.net/musings/page_load_time/, 2010.

[15] KRISHNAMURTHY, B., AND WANG, J. On network-aware clustering of web clients. In *Proceedings of SIGCOMM '00* (New York, NY, USA, 2000), ACM.

[16] KRISHNAN, R., MADHYASTHA, H. V., SRINIVASAN, S., JAIN, S., KRISHNAMURTHY, A., ANDERSON, T., AND GAO, J. Moving beyond end-to-end path information to optimize cdn performance. In *Proceedings of IMC '09* (New York, NY, USA, 2009), ACM.

[17] LEIGHTON, T. Improving Performance on the Internet. *Commun. ACM 52* (February 2009).

[18] NOTTINGHAM, M. Making HTTP Pipelining Usable on the Open Web. http://tools.ietf.org/html/draft-nottingham-http-pipeline-00, 2010.

[19] NYGREN, E., SITARAMAN, R. K., AND SUN, J. The akamai network: a platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev. 44* (August 2010).

[20] OLSHEFSKI, D., AND NIEH, J. Understanding the management of client perceived response time. In *Proceedings of SIGMETRICS '06/Performance '06* (New York, NY, USA, 2006), ACM.

[21] OSTERMANN, S. tcptrace. http://www.tcptrace.org/, 2010.

[22] PADMANABHAN, V. N., AND KATZ, R. H. TCP Fast Start: A Technique For Speeding Up Web Transfers. In *IEEE Globecom* (1998).

[23] QIAN, F., GERBER, A., MAO, Z. M., SEN, S., SPATSCHECK, O., AND WILLINGER, W. Tcp revisited: a fresh look at tcp in the wild. In *Proceedings of IMC '09* (New York, NY, USA, 2009), ACM.

[24] R. STEWART, E. Stream Control Transmission Protocol. RFC 4960, IETF, 2007.

[25] SOUDERS, S. High-performance web sites. *Commun. ACM 51* (December 2008).