Measuring and Optimizing 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], Keith W. Ross[2]

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

Abstract. In this paper, we examine the benefits of split-TCP proxies, deployed in an operational world-wide cloud service network, for accelerating web search queries. We consider a fraction of a network consisting of large number of satellite data centers, which host split-TCP proxies, and a smaller number of large data centers, which compute real time responses to search queries. Detailed measurements reveal that a vanilla TCP splitting solution deployed at the satellite data centers reduces the 95^{th} percentile of latency by as much as 40%, as compared to serving queries directly from the large data centers. Through careful dissection of the measurements, we identify three areas, where optimizations significantly further improve performance: 1) adopting FEC-based low latency reliable protocols between proxies and data centers; 2) choosing proxies based on network latency; 3) specific customizations of TCP for short transfers.

1 Introduction

Cloud Services are delivered with large pools of computational resources that are concentrated in large data centers. Their continued growth, however, critically depends on providing a level of responsiveness comparable with what can be obtained directly from one's user host or with dedicated on-site infrastructure. A key challenge for Cloud Services is to make remote infrastructures appear to end-users as if they were nearby.

Split-TCP proxies [**?**,**?**] can be deployed to improve the responsiveness of Cloud Services. In its simplest form, a split-TCP proxy, deployed close to the end-user, effectively bi-passes TCP *slow start* and reduces the number of round trips by maintaining persistent connections over long-haul links to data centers. Many interactive Cloud Services can benefit from split TCP, including Internet search, Web mail, online document applications, maps, and so on.

Although there are existing deployments of split-TCP proxies in commercial systems (e.g., [**?**,**?**]), there are very few published studies on performance gains based 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 datacenters 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 Internet search as an exemplary case study [**?**], we show that, compared to directly sending queries to data centers, a vanilla TCP splitting solution can reduce 95^{th} percentile latency by 40% . Through detailed analysis, we identify three areas where careful optimization can significantly further improve latency performance. These areas are: (1) adopting FEC-based low latency reliable protocols between the proxy and the data center;(2) choosing proxies based on network latency, instead of geographic distance; and (3) optimizing designs to account for significant observed packet loss between the clients and the proxies. We propose three TCP modifications, for short data transfers (Interactive cloud services in general and WebSearch application in particular). We show that these modifications achieve significant performance improvements, e.g., when we replayed the search queries only with losses we observed that at 95^{th} percentile, the latency is reduced by 24% by our TCP modifications.[4]

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 [**?**]. With millions of incoming queries every day, popular search engines should not only handle large query volumes, but also ensure very rapid response time.

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 compressed 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 data center to complete the actual search and construct the result – as reported on the resultant HTML page.

Figure 1 plots the CDF of response sizes. 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 [**?**]). Figure 2 plots the CDF of response time as observed by the client and the CDF of search time within the data center (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 the client and the data center during the measurement period was

[4] In this paper, we focus on optimizing split-TCP solutions under given proxy locations. We cover an orthogonal and equally important issue – the choice of proxy locations – in an earlier study [**?**] and a companion paper [**?**].

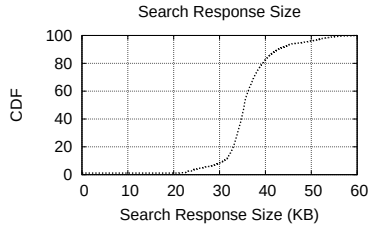

Fig. 1. CDF of response size from a popular search engine

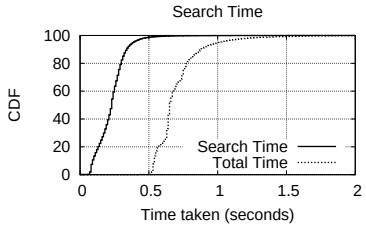

Fig. 2. CDF of time taken by popular search engine for search reply

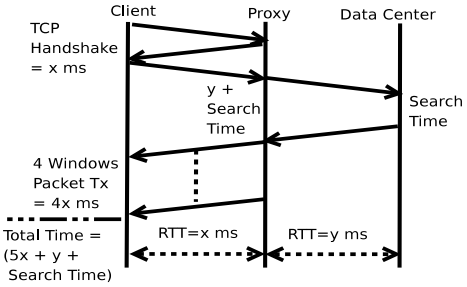

Fig. 3. TCP packet exchange diagram between a HTTP client and a search server with a proxy between them.

around 100 milliseconds. We remark that our measurement 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 data center 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. The client first establishes a TCP connection with the data center server through a three-way TCP handshake. Then, the client sends a 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 (which is a realistic assumption, 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 data center, 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 data center server over the persistent connection with a large TCP window. The data center server processes the search

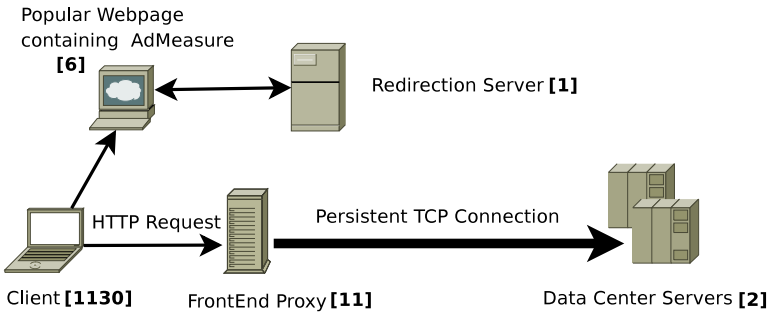

Fig. 4. TCP splitting platform - The [number] beside each component represents the number of that component in our measurement system

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 data center. 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-data center distance, this reduction becomes approximately $4RTT$, which can be quite substantial for interactive applications.

3 Experimental TCP Splitting System

Our experimental system has two major components: a TCP splitting platform and a client measurement platform.

3.1 Measurement System

TCP splitting platform: Our TCP splitting platform consists of three parts: split-TCP proxies, back-end datacenters, and a redirection server [as shown in figure 4]. We deploy our split-TCP proxies [about 2K LOC in C++] on a fraction (11 locations worldwide - XX in US, XX in Europe, XX in Japan, XX) of the satellite datacenters of Microsoft's global distribution and cloud service network. Each proxy perform search queries with the closest (in terms of RTT) Live Search data center. The proxy does not cache any search query results. We first identify the client's geographic location from its IP address, and then redirect it to the closest proxy[5]. The proxy with minimum geographic distance is only an approximation to the proxy with minimum RTT; we will revisit this issue later.

[5] The redirection server incurs overhead for each query. It is a simplified implementation and should be replaced by a DNS server in a production system, where redirection service overhead is amortized over many queries and thus minimal.

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 [**?**] (AdMeasure is implemented in XX Loc in ActionScript). In a nutshell, AdMeasure is deployed as a Flash object, hosted on multiple popular Web pages (e.g., on the homepage of a university). When a client retrieves any one of these Web pages, 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 one workload list from a central AdMeasure server, performs Internet measurements such as issuing search queries to the IPs contained in the workload list, and submits the results back to the AdMeasure server. In our experiments, the AdMeasure server instructs clients (the Flash objects) to issue the search queries to the closest proxy. AdMeasure server contacts the redirection server with IP address of the client, which then returns details of closest proxy to the client. For simplicity, we use a fixed search query term "Barack Obama" with/without proxies. We have verified that when the same query is repeated consecutively directly to the datacenter, the latter ones do not generally finish faster - i.e., there is no caching of search results when queries are issued directly to the datacenter. We have deployed AdMeasure Flash object on multiple popular partner websites (total 6), 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 number of personal homepages.

Each client is instructed to issue six back-to-back queries to the data center through the closest proxy; 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, 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 proxy from 1130 unique clients out of which 952 were located in United States (covering 193 cities). The bias in clients' location originates from the fact that the websites where AdMesure was deployed, were popular mostly in US. 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 real clients. We separate out traces with packet loss in either client to proxy or proxy to datacenter communication (Traces with loss will be re-visited later). From tcpdump outputs on the proxy we identify losses on one path – for simplicity, we assume that retransmissions imply packet losses. However, it is a lot trickier to identify ACK loss, which turns out *not* to be rare. ACK loss can not be directly observed, but can be inferred. Here, we apply a simple heuristic to identify ACK losses – the sequence number gap between all consecutive ACKs is calculated; if

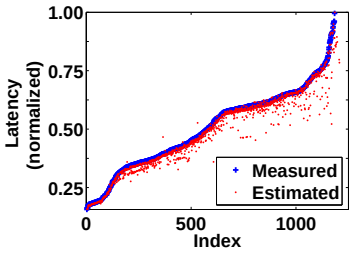

Fig. 5. Latency Model Validation

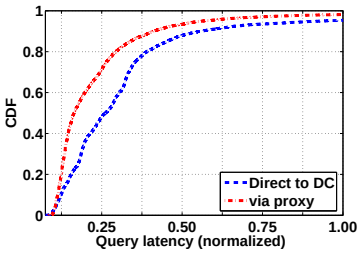

Fig. 6. Gain of TCP Splitting

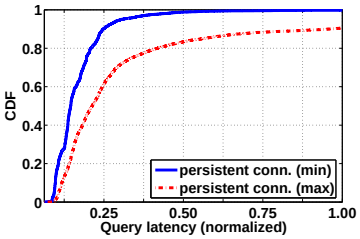

Fig. 7. Impact of Packet Losses

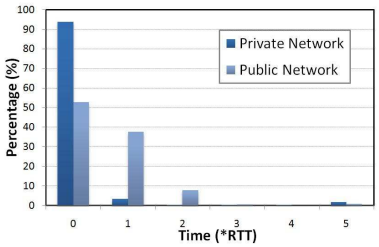

Fig. 8. Time (*RTT) between first and last response packet at proxy from datacenter

the gap is bigger than two MSS (taking into account delayed acknowledgment [**?**]), we assume there is an ACK loss[6].

Considering all traces without packet/ACK loss, we now calculate an estimated latency as: 6 times RTT[7] between client and proxy ($= 6 * x$ - notation follows from figure 3) + 1 RTT between proxy and datacenter ($= y$) + datacenter search time ($= search\_time$). ($y + search\_time$) is obtained directly from tcpdump, as the time from when the proxy forwards the query to the data center 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[8] sorted by measured latency. 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 the proxies are also large. Larger RTTs show more variations, which tend to have a larger impact on inaccuracy.

Impact of Proxy OS: In the course of our experiment, we found that different operating systems exhibit very different TCP window behavior, which in turn has a large impact on latency. Table 1 compares the transmission window for Linux kernel 2.6.20, Windows Server 2003, and Windows Server 2008, using real data collected

[6] Our simple heuristic might over-estimate losses, but is nevertheless conservative for weeding out connections with losses.

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

[8] a few clients visited multiple times

	Linux	Win2003	Win2008
Window Size in Byte			
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

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

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. 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 rest of 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 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 0.995 to 0.60 (both values are normalized) – a savings of 40%!.

Impact of Packet Losses: We instruct each of the 952 clients to issue six queries directly to the datacenter, where the first query is *fresh* and goes through TCP slow-start, the second query is to further warm up the TCP transmission window (simply ignored), and the rest four queries use the established persistent HTTP connection. Figure 7 plots a comparison between a fresh query and the subsequent queries over a persistent connection. For each of the 952 clients, we examine the first query latency (which contains slow-start), the minimum and the maximum latency of the last 4 queries (we do *not* prune cases with loss here). We make the following observations: 1) intuitively, due to TCP slow-start, the first search query experiences much larger latency than the minimum subsequent query; 2) the subsequent queries, despite of the benefit of a persistent HTTP connection and an already opened window, might experience large latency. In the next section, we will design specific measures to mitigate the impact of packet losses. Note that to make comparisons fair, we had remove TCP handshake time from all the results.

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 will occur. 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 typically this is the case for 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 40% of cases take one RTT, indicating that at least one packet loss occurs for these cases. About 5% take two RTTs, indicating multiple packet loss occur for these cases. In both cases, there are non-negligible delays due to packet losses. This suggests that a customized FEC-based low latency reliable protocol (e.g.,[**?**]) between proxy and DataCenter should help. The gain might only be marginal when all the proxies are deployed within the Cloud Service provider's private, well-provisioned network. However, if the provider deploys proxies in third party networks, this so-called "middle-mile" problem[**?**] will become prominent and a customized protocol becomes desirable.

Stress Testing During our measurements we were able to attract over 1000 clients in one week. This rate does not give us an oppurtuinity to measure the performance of our system under load. Specifically, we wanted to measure the search response time between the proxy and the datacenter under load. Nearby queries would suffer the same fate as they go through the same connection between the proxy and the datacenter, ie., if one query suffer loss, queries succeeding it would also suffer due to TCP semantics.

To stress-test our split-TCP platform we did the following on all the 11 proxy locations. We issued back-to-back queries to a webserver on the datacenter to fetch a single image of 50KB size over a single persistent HTTP connection. The web-server was configured to respond to any number of queries on a single persistent HTTP connection. The rate of querying was varied from 1 request/sec to 1000 requests/sec. For each rate, we dispatched queries for 10 seconds and waited for responses for all the queries issued. For every query, we measured the time taken in getting the response.

Figure 9 plots the results of stress testing. For every location we plot one bar for each request rate indicating percent of requests with that requests rate that took 2 more or RTT to deliver the responses. For example, for denmark proxy, at 1 request/sec 20% of the requests took 2 or more RTT. At 1 request/sec rate, we fire a total of 10 requests (in 10 seconds). This means, 2 out of 10 requests took more than 2 RTT. These were the first two requests that warmed up the TCP socket.

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 include that of Padmanaban [**?**] and Mogul [**?**]. Proxies can provide additional benefit through clever techniques, such as using static content to open up TCP window [**?**]. Furthermore, proxies can provide benefit through adaptations of piggyback mechanisms [**?**].

During our experimental deployment, we observed that packet losses are rather common between the clients and the proxies, even though the proxies are deployed on the well-provisioned and well-connected production network. This is a bit surprising,

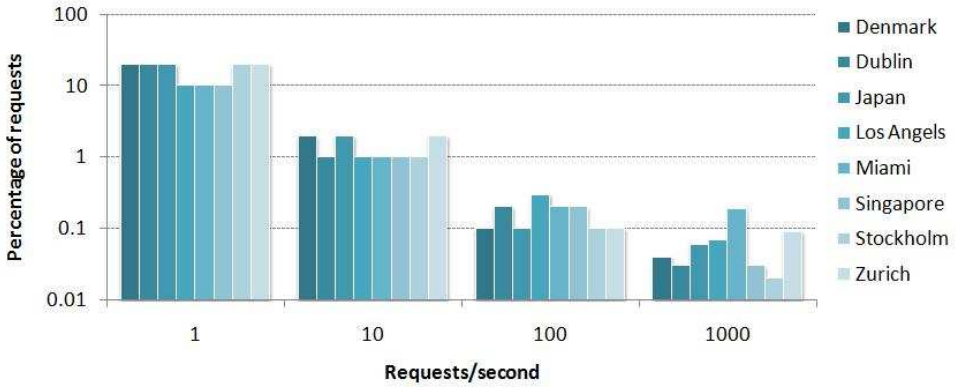

Fig. 9. Stress Test

but yet consistent with observations from other production networks [**?**]. Such reality prompted us to pursue TCP modifications to improve latency performance.

There is a large body of work attempting to improve the TCP protocol, for example, by transmitting the first segment along with the SYN [**?**], by avoiding slow start completely [**?**], by regulating bandwidth at the link layer [**?**], or by dropping duplicate ACK thresholds [**?**]. Our TCP modifications are different. Our focus is to eliminate the potentially large timeouts and avoid congestion avoidance phase in cases where we can infer network situation.

[**?**,**?**] evaluate performance of split-TCP proxies using a very limited set of clients. Through emulation, [**?**] 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. [**?**] estimated the latency penalties incurred by split-TCP proxies. Authors estimated that while application layer split-TCP proxies incur 3.2ms additional delay (using their experimental setup), a kernel level split-TCP implementation incurs only 0.1ms. We focus on very different optimization aspects to reduce latency. Furthermore, our study is based on the measurements from a much larger number of real-world end-users.

5 Future Work And Conclusion

In this paper, we observe that the private network connecting proxies and datacenter servers are loss free most of the time. Also overall loss rate might be lower due to the fact that all the clients, TCP splitting proxies and datacenter servers are from North America, with well developed network infrastructure. Nevertheless, packet loss *does* happen at high percentile. The packet loss could become worse when we move away from North America, e.g., to less “Internet developed” regions of Asia. As a future work, we plan to investigate much broader proxy and datacenter connection scenario, (e.g., connecting proxies in Asia/Europe to datacenters in North America and vice versa), to validate if it is indeed sufficient to rely on persistent TCP to connect proxy and datacenter servers for delay critical applications.

Even within North America, we observe non-trivial packet losses on links connecting proxies to end-users. This has prompted us to examine TCP engine on the proxies. In this paper, we have shown that a few simple modifications can significantly reduce total latency. We plan to further investigate alone this direction and explore other modifications as well.

Finally, 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 [**?**] and plan to conduct large scale measurements in order to answer this question conclusively.

In this paper, we investigate the benefits and optimizations of split-TCP proxies. We measure the real-world benefits of split-TCP proxies by directing over 900 clients to our split-TCP proxies. We show that split-TCP proxies can indeed improve WebSearch performance.