An Argument for Increasing TCP's Initial Congestion Window

Nandita Dukkipati Tiziana Refice Yuchung Cheng Jerry Chu Natalia Sutin
Amit Agarwal Tom Herbert Arvind Jain
Google Inc.
{nanditad, tiziana, ycheng, hkchu, nsutin, aagarwal, therbert, arvind}@google.com

ABSTRACT

TCP flows start with an initial congestion window of at most three segments or about 4KB of data. Because most Web transactions are short-lived, the initial congestion window is a critical TCP parameter in determining how quickly flows can finish. While the global network access speeds increased dramatically on average in the past decade, the standard value of TCP's initial congestion window has remained unchanged.

In this paper, we propose to increase TCP's initial congestion window to at least ten segments (about 15KB). Through large-scale Internet experiments, we quantify the latency benefits and costs of using a larger window, as functions of network bandwidth, round-trip time (RTT), bandwidth-delay product (BDP), and nature of applications. We show that the average latency of HTTP responses improved by approximately 10% with the largest benefits being demonstrated in high RTT and BDP networks. The latency of low bandwidth networks also improved by a significant amount in our experiments. The average retransmission rate increased by a modest 0.5%, with most of the increase coming from applications that effectively circumvent TCP's slow start algorithm by using multiple concurrent connections. Based on the results from our experiments, we believe the initial congestion window should be *at least* 10 segments and the same be investigated for standardization by the IETF.

Keywords

TCP, Congestion Control, Web Latency, Internet Measurements

1. INTRODUCTION AND MOTIVATION

We propose to increase TCP's initial congestion window to reduce Web latency during the slow start phase of a connection. TCP uses the slow start algorithm early in the connection lifetime to grow the amount of data that may be outstanding at a given time. Slow start increases the congestion window by the number of data segments acknowledged for each received acknowledgment. Thus the congestion window grows exponentially and increases in size until packet loss occurs, typically because of router buffer overflow, at which point the maximum capacity of the connection has been probed and the connection exits slow start to enter the congestion avoidance phase. The initial congestion window is at most four segments, but more typically is three segments for standard Ethernet MTUs (approximately 4KB) [5]. The majority of connections on the Web are short-lived and finish before exiting the slow start phase, making TCP's initial congestion window (*init_cwnd*) a crucial parameter in determining flow completion time. Our premise is that the initial congestion window should be increased to speed up short Web transactions while maintaining robustness.

While the global adoption of broadband is growing, TCP's *init_cwnd* has remained unchanged since 2002. As per a 2009 study [4], the average connection bandwidth globally is 1.7Mbps with more than 50% of clients having bandwidth above 2Mbps, while the usage of narrowband (<256Kbps) has shrunk to about 5% of clients. At the same time, applications devised their own mechanisms for faster download of Web pages. Popular Web browsers, including IE8 [2], Firefox 3 and Google's Chrome, open up to six TCP connections per domain, partly to increase parallelism and avoid head-of-line blocking of independent HTTP requests/responses, but mostly to boost start-up performance when downloading a Web page.

In light of these trends, allowing TCP to start with a higher *init_cwnd* offers the following advantages:

(1) Reduce latency. Latency of a transfer completing in slow start without losses [9], is:

$$\lceil log_{\gamma}(\frac{S(\gamma-1)}{init\_cwnd}+1)\rceil * RTT + \frac{S}{C} \tag{1}$$

where S is transfer size, C is bottleneck link-rate, γ is 1.5 or 2 depending on whether acknowledgments are delayed or not, and $S/init\_cwnd \geq 1$. As link speeds scale up, TCP's latency is dominated by the number of round-trip times (RTT) in the slow start phase. Increasing *init_cwnd* enables transfers to finish in fewer RTTs.

(2) Keep up with growth in Web page sizes. The Internet average Web page size is 384KB [14] including HTTP headers and compressed resources. An average sized page requires multiple RTTs to download when using a single TCP connection with a small *init_cwnd*. To improve page load times, Web browsers routinely open multiple concurrent TCP connections to the same server. Web sites also spread content over multiple domains so browsers can open even more connections [8]. A study on the maximum number of parallel connections that browsers open to load a page [15] showed Firefox 2.0 opened 24 connections and IE8 opened 180 connections while still not reaching its limit. These techniques not only circumvent TCP's congestion control mechanisms [13], but are also inefficient as each new flow independently probes for end-to-end bandwidth and incurs the slow start overhead. Increasing *init_cwnd* will not only

[0]Paper is under review for publication.

mitigate the need for multiple connections, but also allow newer protocols such as SPDY [1] to operate efficiently when downloading multiple Web objects over a single TCP connection.

(3) Allow short transfers to compete fairly with bulk data traffic. Internet traffic measurements indicate that most bytes in the network are in bulk data transfers (such as video), while the majority of connections are short-lived and transfer small amounts of data. Statistically, on start-up, a short-lived connection is already competing with connections that have a congestion window greater than three segments. Because short-lived connections, such as Web transfers, don't last long enough to achieve their fair-share rate, a higher *init_cwnd* gives them a better chance to compete with bulk data traffic.

(4) Allow faster recovery from losses. An initial window larger than three segments increases the likelihood that losses can be recovered through Fast Retransmit rather than the longer initial retransmission timeout. Furthermore, in the presence of congestion, the widespread deployment of Selective Acknowledgments (SACK) enables a TCP sender to recover multiple packet losses within a round-trip time.

We propose to increase TCP's *init_cwnd* to *at least* ten segments (approximately 15KB).[1] To that end, we quantify the latency benefits and costs, as measured in large scale experiments conducted via Google's front-end infrastructure serving users a diverse set of applications.

Ideally, we want to pick an *init_cwnd* satisfying each of the following properties: *(i)* minimize average Web page download time; *(ii)* minimize impact on tail latency due to increased packet loss, and *(iii)* maintain fairness with competing flows. Raising the initial congestion window to ten segments can reasonably satisfy these properties. It improves average TCP latency, yet is sufficiently robust for use on the Internet. In the following, we articulate the pertinence of ten segments to TCP's initial congestion window:

(1) Covers $\approx$90% of HTTP Web objects: 90% of HTTP responses from the top 100 and 500 sites fit within 16KB [14], as shown in the response size distribution of Figure 1. The distribution also shows that about 90% of Google Web search, Maps, and Gmail responses fit in about 15KB or ten segments; for applications with larger average size such as Photos, 20-30% more responses can fit within ten segments as compared to three segments.

(2) Improves latency, while being robust: Figure 2 shows the average TCP latency for Google Web search as *init_cwnd* is varied from 3 to 42 segments, in a small scale experiment conducted concurrently on six servers in the same data center (all offered similar traffic load and user base). Using ten segments improves the average TCP latency compared to using three segments. We note that raising *init_cwnd* to 16 improves latency further. However, much larger values, such as 42, show a degradation in latency, likely due to increased packet losses. Larger scale experiments described in the rest of the paper demonstrate at length the benefits and potential costs of using an initial congestion window of ten segments.

There are numerous studies in literature on speeding up short transfers over new TCP connections [10]. These techniques range from faster start-up mechanisms using cached congestion windows such as TCP Fast Start and Congestion

[1]We assume IP over Ethernet and a maximum segment size of 1430 bytes.

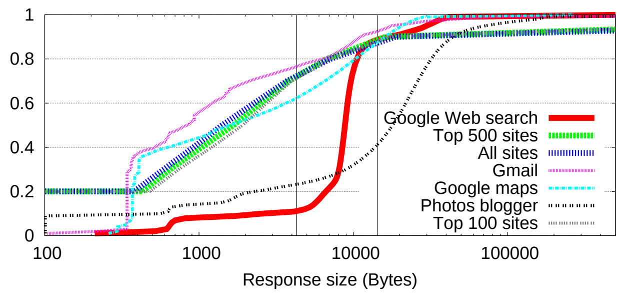

Figure 1: CDF of HTTP response sizes for top 100 sites, top 500 sites, all the Web, and for a few popular Google services. Vertical lines highlight response sizes of 3 and 10 segments.

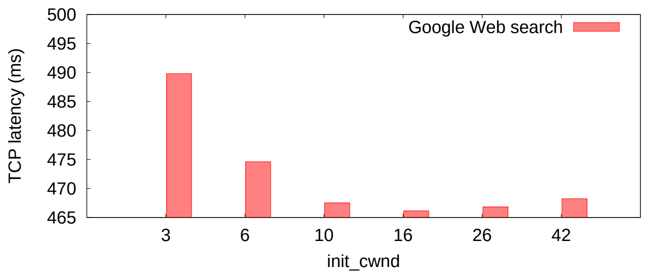

Figure 2: TCP latency for Google search with different *init_cwnd* values.

Manager to more complex schemes requiring router support such as Quick Start. These solutions are neither widely deployed, nor standardized, and do not have practical reference implementations.

The rest of the paper is arranged as follows: Section 2 describes the experiments' setup and datasets. Section 3 presents an analysis of receiver advertised windows. Section 4 describes the experiment results with an initial congestion window of ten, quantifying its benefits and cost analyzed by network properties (bandwidth, BDP, RTT), as well as traffic characteristics. Section 5 concludes the paper with a discussion on future work.

2. EXPERIMENT SETUP AND DATASETS

Our experiments consist of enabling a larger initial congestion window on *front-end servers* in several data centers at geographically diverse locations. We compare the results using the larger window against data from the same data centers using the standard initial congestion window as a baseline. The front-end servers terminate TCP connections to users on the Internet, and they may communicate with some number of internal back-end servers which compute and provide response data for Google applications. The front-end servers run Linux with a reasonably standards compliant TCP implementation [6] (the congestion control algorithm used is TCP CUBIC), and the initial congestion window is configured using the `initcwnd` option in the `ip route` command. All front-end servers within a data center are configured with the same initial congestion window. The network is set up such that users in the same BGP subnet are served from the same data center, and if a service

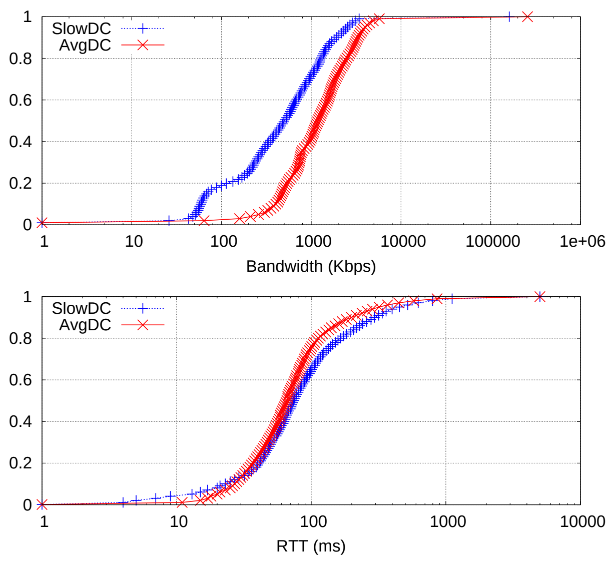

Figure 3: CDF of response bandwidth and RTT.

requires multiple connections they are all served from the same data center; thus all connections to Google services from a particular subnet (at a /24 level) on the Internet are subject to the same initial congestion window on the server for the duration of the experiment.

We present data for two representative data centers denoted *AvgDC* and *SlowDC*. AvgDC serves subnets with average connection bandwidths that are similar to the worldwide average reported in [4], with a median bandwidth of 1.2Mbps, and about one third of the traffic to subnets with bandwidths >2Mbps. SlowDC serves subnets with a larger proportion of lower connection bandwidths, with a median bandwidth of 500Kbps, and nearly 20% of the traffic to subnets with bandwidths <100Kbps. The median RTT for traffic is approximately 70ms in both data centers. Figure 3 shows the distribution of bandwidth and RTT for traffic served by AvgDC and SlowDC.

We collected data from each data center over two consecutive weeks. During the first week, the front-end servers were configured with an initial congestion window of ten (the experiment period). In the second week, the servers used the standard initial congestion window of three (the baseline period). The front-end servers logged information for a sample percentage of HTTP transactions (requests and corresponding responses). Each log entry records the client and server IP addresses, the minimum TCP RTT seen on the connection, the number of unique bytes sent, the number of retransmitted bytes, and the Google application that provided the response. Several timestamps are logged for each transaction which are denoted by T_* in Figure 4; of particular interest are T_{Res1}, the time at which the server sends the first byte of a response, and T_{ACK}, the time when the server receives the last acknowledgment for the response. The TCP latency for a transaction, which we expect to be affected by changing the initial congestion window, is $T_{ACK} - T_{Res1}$. Averaging this across all logged responses provides the average latency for a test configuration in the experiment. The latency impact between the experiment period and the baseline period is $\text{AvgLatency}_{Base} - \text{AvgLatency}_{Exp}$, and the percentage difference is
$(\text{AvgLatency}_{Base} - \text{AvgLatency}_{Exp}) * 100/\text{AvgLatency}_{Base}$.

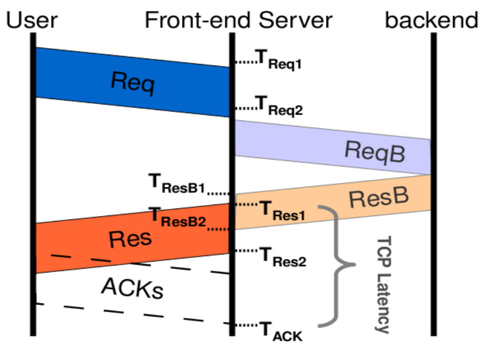

Figure 4: Latency timeline of an HTTP request/response.

Dataset	# Subnets	# Response	Volume (TB)
AvgBaseData	1M	5.5B	39.3
AvgExpData	1M	5.5B	39.4
SlowBaseData	800K	1.6B	9.3
SlowExpData	800K	1.6B	9.1

Table 1: Datasets presented in the paper.

We are also interested in the metric of retransmission rate, defined as the ratio of retransmitted bytes to the unique bytes transmitted.

It should be be noted that with persistent connections multiple HTTP transactions can be transferred over a single connection, where the latter transactions on the connection may benefit from typical congestion window growth; we do not isolate such transactions in the data. Also, data are only collected from the server side of a connection, which limits our perspective and allows only a temporal comparison as opposed to a fully controlled experiment. The front-end servers only log TCP traffic; however, according to [12], this is not a significant limitation.

Table 1 lists our data sets collected in AvgDC and SlowDC with the number of /24 subnets, responses, and volume of data served during the experiment period and the baseline period.

3. CLIENT RECEIVE WINDOWS

Since TCP can only send the minimum of the congestion window and the client's advertised receive window, the receive window (*rwnd*) may limit the potential performance improvement of increasing *init_cwnd*. In other words, clients

rwnd of first HTTP request		
OS	% > 15KB	Avg.
FreeBSD	91%	58KB
iPhone	66%	87KB
Linux	6%	10KB
Mac	93%	270KB
Win 7	94%	41KB
Win Vista	94%	35KB
Win XP	88%	141KB

Table 2: Initial receive window sizes. The operating system is extracted from the HTTP User-Agent.

	AvgDC			SlowDC		
Qtls	Exp	Base	Diff [%]	Exp	Base	Diff [%]
Avg	514	582	68 [11.7]	751	823	72 [8.7]
10.0	174	193	19 [9.84]	204	211	7 [3.32]
50.0	363	388	25 [6.44]	458	474	16 [3.38]
90.0	703	777	74 [9.52]	1067	1194	127 [10.64]
95.0	1001	1207	206 [17.07]	1689	1954	265 [13.56]
99.0	2937	3696	759 [20.54]	5076	5986	910 [15.20]
99.9	8463	10883	2420 [22.24]	16091	18661	2570 [13.77]

Table 3: Latency quantiles (in ms) and improvement for Web search. Experiment (*init_cwnd*=10) and baseline results are referred to as *E*xp and *B*ase respectively. *D*iff is the difference between baseline and experiment.

need to advertise at least a 15KB receive window on a connection to fully benefit.

To evaluate the extent of this issue, we inspected the *rwnd* field in the TCP header of the first HTTP request of the clients' connections to front-ends at geographically distributed locations for 24 hours. Table 2 shows that operating systems such as Windows, MacOS, and FreeBSD specify large *rwnd*s in the first HTTP request. The major exception is Linux which mostly uses 8KB.[2] Overall, more than 90% of client connections have a large enough receive window to fully benefit from using *init_cwnd*=10 segments.

4. EXPERIMENT RESULTS

In this section we discuss the latency benefits and costs we observed in our experiments with a larger initial congestion window.

As most observations are common across the experiments, we first present results from AvgDC and where results materially differ or deserve attention for low bandwidth subnets, we discuss data of SlowDC. Similarly, we present data from Google Web search queries, but where necessary also discuss the performance of applications with different characteristics, such as Google Maps that serves content via concurrent TCP connections, and iGoogle or Blogger photos that have relatively large average response sizes as shown in Figure 1.

4.1 Impact on Reducing TCP Latency

The average latency improvement of Web search in AvgDC is 11.7%(68ms) and in SlowDC 8.7% (72ms). Table 3 shows the improvement across quantiles from 10% to 99.9%. The higher quantiles show relatively larger improvements, likely due to the benefits of a large *init_cwnd* in high RTT networks, e.g., saving two RTTs over a 300ms network path improves latency by 600ms.

In the following discussion, we present further analysis of the experimental data as functions of traffic characteristics and subnet properties of bandwidth (BW), round-trip time (RTT), and bandwidth-delay product (BDP).

Impact on subnets of varying BW, RTT and BDP

Figure 5 compares the latency improvements across different buckets of subnet BW, RTT, and BDP. The subnet bandwidth for each response is looked up from a central repository of bandwidth estimates to worldwide /24 subnets.[3] RTT is the minimum recorded packet round-trip time by each response, including that of SYN/SYN-ACK, and is the closest estimate of path propagation delay. BDP is the product of RTT and BW.

The average response latency improved across all buckets of subnet properties. As expected, the largest benefits are for high RTT and high BDP networks, because most responses can fit within the *pipe*, i.e. response size $<= RTT_{high} * BW$, and can finish within one RTT, as opposed to lasting multiple RTT rounds in slow start. We note that in our sample set, high BDP is mostly a consequence of high RTT paths.

Correspondingly, responses from low RTT paths experience the smallest improvements; not just in absolute numbers, but also in terms of percentage. The percentage im-

[2]A pending patch submitted to the Linux kernel allows users to configure initial *rwnd*.

[3]The central repository keeps track and computes running averages of achieved throughput to user subnets using logs collected by all the Google servers, and is updated on a monthly basis.

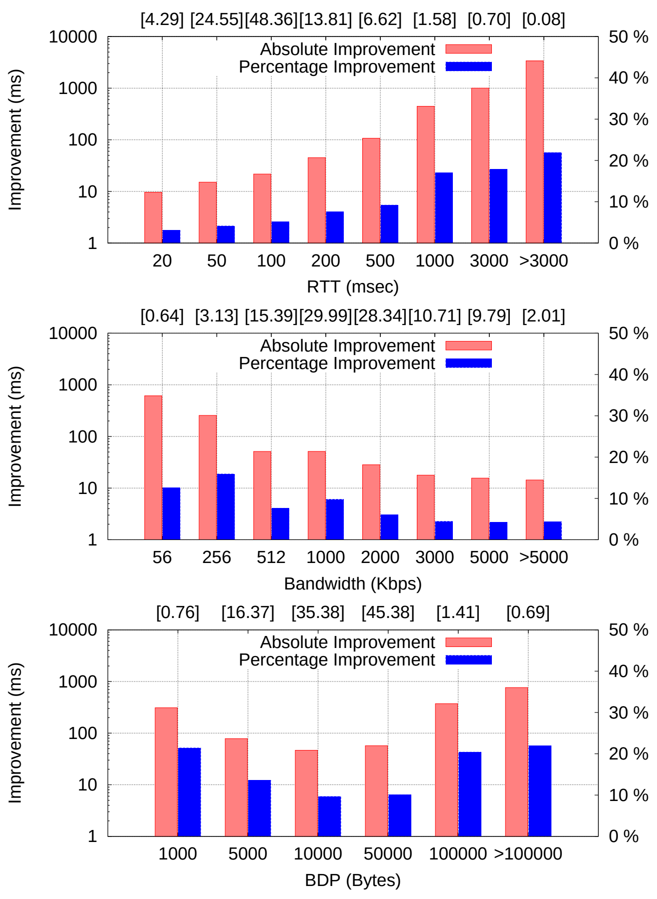

Figure 5: Average response latency for Web search bucketed by RTT, BW, and BDP at AvgDC. The left y-axis shows the absolute latency improvement (in ms), and the right y-axis shows the percentage improvement. The buckets are represented in the lower x-axis scale, while the numbers at the top of the plot represent the percentage responses in each bucket.

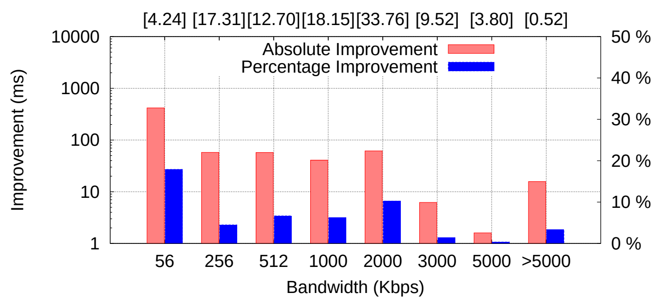

Figure 6: Average response latency for Web search bucketed by BW at SlowDC.

provement is smaller because a smaller fraction of responses benefit in low RTT networks i.e., responses with size $<= RTT_{low} * BW$. Those responses with size $> RTT_{low} * BW$ need multiple RTTs regardless of the initial congestion window.

Figure 5 shows that latency also improves across the range of subnet bandwidths. At larger bandwidths ($\geq$2Mbps) the improvements in both absolute and percentage are approximately the same. This is expected because TCP's latency during slow start in the absence of losses is given by:

$$\text{Slow start latency} = N_{Slowstart} * RTT + \frac{size}{BW} \qquad (2)$$

where $N_{Slowstart}$ is the number of round trips in slow start. In high BW networks the transmission delay is a relatively small component and the improvement is in the number of RTTs in the slow start phase, which is independent of BW.

We note that the percentage improvements shown in Figure 5 underestimate the attainable latency improvements. For Google Web search, Equation 2 has an additional component of back-end latency ($\approx$200ms) before the entire HTTP response can be made available at the front-end (at epoch T_{ResB2} in Figure 4). This is because of the latency involved at the back-end between sending an initial burst of three to four packets consisting of the response headers, and delivering the rest of the response body. The back-end processing time manifests as idle time on the connection, lowering the achievable percentage improvement.

Low bandwidth subnets

Responses from low bandwidth subnets demonstrated significant latency improvements as shown in the 0-56 and 56-256Kbps buckets of Figure 5. This is also true in SlowDC, as shown in Figure 6, which has an even larger portion of traffic in low BW buckets – approximately 4% and 17% of Web search traffic in buckets 0-56Kbps and 56-256Kbps respectively. A large portion of this traffic is from dial-up modems as well as low bandwidth mobile networks. We note that the improvement in low BW buckets is also reflected in low BDP buckets. There are two reasons why low BW networks

Qtls	10	50	90	95	99
Exp	218	660	3023	5241	12328
Base	261	774	3997	6724	14374
Diff	43	114	974	1483	2046

Table 4: Latency quantiles (in ms) for Web search from subnets with bandwidth $\leq$56Kbps in SlowDC.

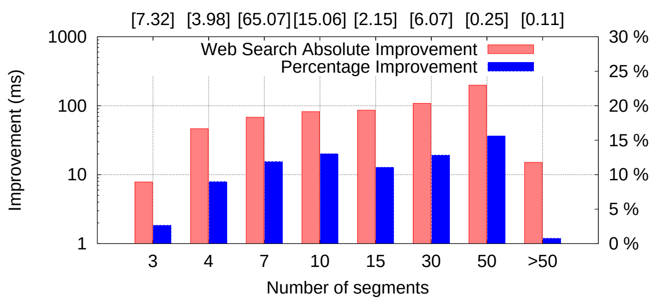

Figure 7: Average response latency at AvgDC for Web search bucketed by number of segments. The left y-axis shows the absolute improvement (in ms), and the right y-axis shows the percentage improvement. The buckets are represented in the lower x-axis scale, while the numbers at the top of the plot represent the percentage responses in each bucket.

observe improved latency:

(i) Fewer slow start rounds: Unlike many large BW networks, low BW subnets with dial-up modems and certain mobile networks [11] have inherently large RTTs, probably a consequence of coding techniques used at the link layer. In our measurements, the average of the minimum RTTs for 0-56 and 56-256Kbps subnets is 343ms and 250ms at SlowDC. A larger initial window can save an RTT even on slow links; e.g., with an RTT of 500ms, an 8KB transfer with *init_cwnd*=10 can complete within a round in slow start on a 128Kbps link, but would otherwise take at least two rounds.

(ii) Faster loss recovery: An *init_cwnd* larger than three segments increases the chances of a lost packet to be recovered through Fast Retransmit as opposed to being recovered by the longer initial retransmission timeout [7]. The fast retransmit algorithm requires three duplicate acknowledgments as an indication that a segment has been lost rather than reordered. A larger *init_cwnd* allows a TCP sender to receive better and faster feedback via more acknowledgments.

We conclude the discussion for low BWs with Table 4 that shows latency quantiles for traffic in 0-56Kbps for SlowDC. We note that latency improves not only in the median but also for higher quantiles.

Impact on traffic of varying sizes of responses

Figure 7 illustrates the improvement for responses of varying sizes for Web search traffic (average size: 10.5KB).

Responses greater than three segments naturally demonstrate larger improvements, because they experience fewer number of round-trips in slow start as compared to the baseline. The RTT rounds saved, and therefore the improvement in absolute latency, grows (up to a limit) with response sizes.[4] We note that the theoretical number of round-trips in slow start which can be saved has an upper bound of $\lceil log_{\gamma}(\frac{init\_cwnd_{high}}{init\_cwnd_{low}})\rceil$ (computed from Equation 1). This yields a saving of up to three rounds for γ=1.5 (with delayed acks), $init\_cwnd_{high}$=10 and $init\_cwnd_{low}$=3. In practice the savings can be up to four RTTs. The difference arises from the fact that Equation 1 rounds up a fractional increase in congestion window per RTT which does

[4]Note that the last bucket (>50 segments) in Web search latency has a small sample number.

	AvgDC			SlowDC		
Qtls	Exp	Base	Diff	Exp	Base	Diff
10.0	301	317	16	299	302	3
50.0	421	450	29	503	517	14
90.0	943	1060	117	1037	1161	124
95.0	1433	1616	183	1451	1627	176
99.0	3983	4402	419	3535	3979	444
99.9	9903	11581	1678	10471	10775	304

Table 5: Per-subnet average latency quantiles (in ms) for Web search at AvgDC and SlowDC.

not occur in practice. Because larger responses take longer to complete, the percentage improvement in completing the flow does not grow by the same factor as the absolute benefits.

Responses with size $\leq$3 segments

Response sizes of three segments and smaller also demonstrate small latency benefits (< 10ms). This can occur because a connection starting with *init_cwnd*=10 is more likely to have a larger *cwnd* for subsequent responses in the same connection even after losses (upon a loss the congestion window is reduced by half of its current size).

Results from other Google applications also show that responses with sizes $\leq$ 3 segments perform no worse in the experiments with *init_cwnd*=10 as compared to the baseline. This may be evidence that connections with larger *init_cwnd*s are not detrimental to ones having smaller congestion windows.

Impact on per-subnet latency

We can statically break out subnets being served by 24-bit IPv4 address prefixes (/24 subnets). A /24 subnet is an approximation to a user base that is likely offered a similar set of services by a common ISP. In this section, we discuss how these subnets perform in terms of average latency.

From our experimental data, we averaged the latency of Web search responses per /24 subnet. Only the subnets observed in both the baseline and experiment are considered; these subnets accounted for >99% of the traffic in our data sets. We then computed the distribution of per-subnet latency averages.

Table 5 shows that average subnet latency improves across the different quantiles. Note that the absolute per-subnet latencies for baseline and *init_cwnd*=10 in each quantile are higher than the corresponding quantile in Table 3; this is expected as there is usually greater traffic volume originating from subnets with higher bandwidths.

Impact on applications with different characteristics

So far we have discussed only Web search latency. Other applications can generate traffic with different characteristics either because of relative larger response size, such as iGoogle (avg. size: 28KB), or because they serve content using multiple concurrent TCP connections, such as Maps — browsers typically open four to six connections per domain and Google's Maps uses two domains to download tiles, making up to twelve active concurrent connections. Table 6 shows that latency improves across different quantiles even for Maps and iGoogle.

Sec. 4.2 further discusses Maps behavior in the SlowDC.

Qtls	Maps [%]	iGoogle [%]
10.0	1 [2.1]	35 [21.6]
50.0	5 [2.2]	77 [20.3]
90.0	26 [3.8]	187 [17.8]
95.0	36 [3.1]	339 [18.9]
99.0	95 [3.1]	854 [14.4]
99.9	278 [3.6]	2231 [11.9]

Table 6: Latency quantiles improvement (in ms) for Google Maps and iGoogle in AvgDC. Their average response size is, respectively, 6.4KB and 28KB.

4.2 Negative Impact

In previous sections, we have quantified the overall benefits of using a higher *init_cwnd*; in this section, we discuss its costs, specifically cases where latency increases. Increase in latency primarily arises from packet losses caused by overflowing bottleneck buffers, either at end-systems or at intermediate routers and switches. Losses prolong TCP flows by adding extra RTTs required to recover lost packets, and occasionally even resulting in retransmission timeouts.

Internet measurements and studies show that a critical bottleneck in the Internet lies in the last mile at the user's access link [3]. Thus, if there is a cost associated with *init_cwnd*=10, it is likely that we will observe increased congestion and packet losses in the experiment traffic. In this section, we quantify this effect.

The effect on retransmission rate

TCP's retransmission rate represents an upper bound for the percentage of packets lost due to congestion. In this section, we quantify the impact of *init_cwnd*=10 on the retransmission rate of different applications. The average retransmission rate is defined as the ratio of retransmitted bytes to that of unique bytes transmitted, expressed as a percentage.

Table 7 summarizes the overall and per-service retransmission rate for AvgDC and SlowDC. The overall increase with *init_cwnd*=10 in AvgDC is <0.5% and in SlowDC is <1%. Naturally, the increase in retransmission is not uniform across applications, but concentrated around applications serving content on multiple concurrent TCP connections, and/or having large average response sizes. In both cases, with a higher initial congestion window the effective burst size transmitted is larger, resulting in increased re-

AvgDC				
	All	Web search	Maps	Photos
Exp	2.29 [6.26]	1.73 [5.63]	4.17 [7.78]	2.64 [11.14]
Base	1.98 [6.24]	1.55 [5.82]	3.27 [7.18]	2.25 [10.38]
Diff	0.31 [0.02]	0.18 [-0.20]	0.90 [0.60]	0.39 [0.76]
SlowDC				
Exp	4.21 [8.21]	3.50 [10.44]	5.79 [9.32]	6.10 [22.29]
Base	3.54 [8.04]	2.98 [10.17]	3.94 [7.36]	4.97 [19.99]
Diff	0.67 [0.17]	0.52 [0.26]	1.85 [1.97]	1.12 [2.30]

Table 7: Average retransmission percentage for services in AvgDC and SlowDC. Each entry presents percentage retransmission rate and the percentage of the responses with >0 retransmissions (within brackets).

Maps			
Qtls	Exp	Base	Diff [%]
10.0	19	27	8 [29.6]
50.0	170	176	6 [3.4]
90.0	647	659	12 [1.8]
95.0	1172	1176	4 [0.3]
96.0	1401	1396	-5 [-0.4]
97.0	1742	1719	-23 [-1.3]
99.0	3630	3550	-80 [-2.3]
99.9	10193	9800	-393 [-4.0]

Table 8: Latency quantiles (in ms) of Maps at SlowDC.

transmission rates.

We also note that overall in both AvgDC and SlowDC, the percentage of responses experiencing retransmissions has only increased by a small portion — an increase of 0.02% in AvgDC and 0.17% in SlowDC.

Applications using concurrent TCP connections

Traffic patterns from applications using multiple concurrent TCP connections with a large *init_cwnd* represent one of the worst-case scenarios where latency can be adversely impacted by bottleneck buffer overflow. As an example, the use of four to six connections by browsers per domain, coupled with the common practice of Web sites using two subdomains makes the *effective init_cwnd* between 80-120 segments for certain applications such as Google Maps. Table 6 shows that this does not cause a noticeable degradation in AvgDC.

In SlowDC, Table 8 shows that while Maps has improved its median latency, latencies in higher quantiles such as 96^{th} and above have experienced a degradation relative to the baseline.

There is no evidence from our measurements that this latency degradation is well correlated to low bandwidth subnets. We conjecture that such impact is not a consequence of low bandwidth alone, but other factors working in concert such as the amount of buffering at a bottleneck link, and the offered load at the link i.e., the number of simultaneous flows multiplexed and their usage patterns.

5. CONCLUSIONS

Increasing TCP's initial congestion window is a small change with a significant positive impact on Web transfer latency. While the numerous studies in literature to speed up short transfers may be viable solutions in the future, none are deployed or standardized today. In contrast, a far simpler solution of increasing TCP's initial congestion window to a value commensurate with current network speeds and Web page sizes is practical, easily deployable, and immediately useful in improving Web transfer latency.

In the longer term, a larger initial congestion window will also mitigate the need for applications to use multiple concurrent connections to increase download speed. Based on our large scale experiments, we are pursuing efforts in the IETF to standardize TCP's initial congestion window to *at least* ten segments. Preliminary experiments with even higher initial windows show indications of benefiting latency further while keeping any costs to a modest level. Future work should focus on eliminating the initial congestion window as a manifest constant to scale to even large network speeds and Web page sizes.

6. ACKNOWLEDGMENTS

We would like to thank John Reese for conducting, monitoring and coordinating the many large scale experiments over the past several months. We would also like to extend our gratitude to Laurent Chavey, Mike Belshe, Roberto Peon and Robert Love for their insights and helpful discussions. We are extremely grateful to Urs Hölzle for giving us the possibility of conducting this work.

7. REFERENCES

[1] SPDY: An experimental protocol for a faster web. http://dev.chromium.org/spdy, 2009.

[2] AJAX - Connectivity Enhancements in Internet Explorer 8. http://msdn.microsoft.com/en-us/library/cc304129(VS.85), 2010.

[3] Akamai. Internet Bottlenecks. http://www.akamai.com/html/perspectives/whitepapers_content.html, 2000.

[4] Akamai. The State of the Internet. 3rd Quarter 2009. http://www.akamai.com/stateoftheinternet, 2009.

[5] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. RFC 3390, 2002.

[6] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681, 2009.

[7] M. Allman, V. Paxson, and J. Chu. Computing TCP's Retransmission Timer. Internet-draft draft-paxson-tcpm-rfc2988bis-00, work in progress, 2010.

[8] M. Belshe. A Client-Side Argument For Changing TCP Slow Start. http://sites.google.com/a/chromium.org/dev/spdy/, 2010.

[9] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. In *Proceedings of IEEE Infocom*, 2000.

[10] J. Iyengar, A. Caro, and P. Amer. Dealing With Short TCP Flows: A Survey of Mice in Elephant Shoes. In *Tech Report, CIS Dept, University of Delaware*, 2003.

[11] P. Benko and G. Malicsko and A. Veres. A large-scale, passive analysis of end-to-end TCP performance over GPRS. 2004.

[12] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger. TCP Revisited: a Fresh Look at TCP in the Wild. In *IMC*, 2009.

[13] S. Floyd, and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. In *IEEE/ACM Transactions on Networking*, 1999.

[14] S. Ramachandran, and A. Jain. Web page stats: size and number of resources. http://code.google.com/speed/articles/web-metrics.html, 2010.

[15] S. Souders. Roundup on Parallel Connections. http://www.stevesouders.com/blog/2008/03/20/roundup-on-parallel-connections/, 2008.