Impact of TCP Variants on HTTP Performance

K. Farkas[1], P. Huang[1], B. Krishnamurthy[2], Y. Zhang[2], J. Padhye[3]

I. Introduction and Motivation

For nearly seven years Web traffic has been a majority of traffic on the Internet. Virtually all implementations of HTTP use TCP as the transport layer. During the evolution of the HTTP/1.1 version of the protocol, several of the performance problems related to the Web were traced to HTTP's reliance on TCP. TCP was optimized for long transfers but most Web responses consists of an average of 8 Kilobytes that fit in a handful of packets. The setting up and tearing down cost of TCP could dominate the overall cost of a HTTP transfer. Rather than requiring a separate TCP connection for each of the resources requested by a browser, persistent connections were introduced in HTTP/1.1 to reduce overhead and latency and this has been borne out by studies (e.g., [1]).

Given the variety of TCP variants in use (NewReno, Reno, Tahoe, etc.), it is a natural question to examine which specific aspects of TCP affect Web performance and if certain variants help provide better Web performance. By Web performance, we mean the end-to-end latency experienced by Web clients and the impact on the Web server in handling the requests.

There are four phases to our work:

1. We first provide a taxonomy of what impacts Web performance from the viewpoint of TCP and HTTP. Towards this end, we identify the key TCP as well as HTTP parameters that need to be studied.
2. We then create an experimental methodology to study the various key facets identified above.
3. An *ns* based simulation that uses a reasonable workload to examine precisely how the various parameters actually affect Web transfers is carried out.
4. A live measurement on typical downloading of Web traffic is then carried out using a modified user-level TCP stack in conjunction with an actual Web server to ensure coverage not often available via simulation.

Ideally, our goal is to tell a Web proxy or a server that given their traffic mix, how performance would vary if their TCP stack or the parameter choices varied.

The rest of the paper is divided as follows: We examine the set of TCP and HTTP parameters, as well as the interaction between the two layers in Section II. Section III describes the simulation environment, the software used as well as the results of the simulation carried out. Next, Section IV describes the live experiment carried out. Section V summarizes the different results in the simulation and the live experiment, and we conclude with a look at the work that remains to be done.

[1]ETH Zürich, {farkas, huang}@tik.ee.ethz.ch; [2]AT&T Labs-Research, {bala, yzhang}@research.att.com; [3]ACIRI, padhye@aciri.org

II. TCP and HTTP Parameters that Influence Web Performance

In this section we sketch the design space of the pertinent TCP and HTTP parameters that affect Web performance.

We start with the TCP-only parameters, then examine the HTTP-specific parameters, and finally examine how the two layers interact. The design space guides the simulation and the live experiments.

A. TCP Parameters that Influence Web Performance

Several variants of the basic TCP algorithm have been developed. Each variant contains several parameters. Here, we enumerate various TCP parameters that influence the performance of a Web connection.

TCP variants: There are several variants of the basic TCP congestion control algorithm. In this paper we consider three main variants: Tahoe [2], Reno [2] and NewReno [2]. Many TCP stacks also offer the use of Selective Acknowledgment (SACK) option, which helps to speed up a Web transfer. We examined the impact of SACK as well.

Maximum segment size: The maximum segment size (MSS) of a TCP connection is the maximum amount of data the server may send in a TCP packet. With larger MSS, there is less per-packet overhead of administrative information. However, the average value of congestion window of the sender will decrease, given a larger MSS. This increases the probability that packet losses will lead to timeout, thereby reducing the throughput of the TCP connection.

Delayed Acknowledgment: When a TCP receiver uses delayed acknowledgment, it usually sends one acknowledgment for every two data packets it receives. This strategy reduces the number of packets sent into the network. However, this also slows down the rate of growth of the congestion window of the sender and reduces the sender throughput.

Initial Retransmission Timeout: A TCP sender maintains an estimate of the round trip time (RTT) to the receiver. This estimate is used to set the retransmission timer. A smaller value is better for the throughput of the connection,

but may inject some extra packets in the network. A high RTO implies that lost packets will not be detected quickly.

Timer granularity: TCP uses an internal timer to calculate RTT and RTO. The granularity of the timer determines the accuracy with which RTT and RTO are calculated, and may affect the throughput of a TCP (and thus HTTP) connection.

Initial congestion window size: The initial congestion window (ICW) size of a TCP sender is the amount of data the sender is allowed to send at the beginning of the connection. A large ICW can reduce the user-perceived latency. On the other hand, large ICW results in more bursty network traffic, resulting in increased loss and delay.

Receiver buffer sizes: The amount of data a TCP connection may have outstanding in the network is limited by the receiver's advertised window size, which is the amount of buffer the receiver has allotted for this TCP connection.

Timestamp: Use of timestamps may increase the accuracy of RTO calculations and so we examined the overall role played by the timestamp option on Web performance.

B. HTTP Parameters that Influence Web Performance

Web performance consists of three key issues: user perceived latency, server load, and network load. Accordingly, we enumerated the various HTTP parameters, sorted them in decreasing order of their contribution to Web performance and selected those that were likely to interact with the transport layer.

We thus focus on connection management. In connection management we consider parallel connections, persistent connections, and pipelining. In the earlier versions of the HTTP protocol, a separate TCP connection was required to download each resource. Even before the introduction of persistent connections it was common for Web browsers to establish multiple parallel connections to the server and fetch the images in parallel.

A HTTP persistent connection allows the already established connection to stay open beyond a single request-response exchange. Persistent connections reduce the need for multiple TCP connections to be set up and torn down, avoiding the high overhead for the typically short Web transfers. If multiple requests were sent on the same persistent connection without waiting for individual responses from the server, latency could be further reduced. Such *pipelining* avoids the roundtrip delay of waiting for each of the response before sending additional requests.

Overall, these connection management schemes can improve Web performance by reducing page download time, avoiding connection setup time, and avoiding request time. They can also have negative impact when interacting with TCP. Next we discuss how the connection management schemes interact with TCP and where negative and additional positive impact can arise from the interactions.

C. TCP and HTTP Interaction

HTTP and TCP interact in a number of ways. We focus our attention on the critical parameters of TCP and HTTP.

Network condition: Depending on the network condition, the use of parallel connections, persistent connections with or without pipelining, could result in the underlying TCP experiencing a different degree or distribution of packet drops. Since the different TCP variants (Reno, NewReno, and SACK, etc.) react differently to packet drops, the overall Web performance could vary.

Persistent Connections: The use of persistent connections avoids establishing TCP connections for each Web resource downloaded. Its effect is likely depending on the initial RTO and timer granularity.

Parallel Connections: The use of parallel connections and persistent connections (with or without pipelining) could result in a larger aggregated window size. In case of packet drops, the aggregated transfer could experience a more aggressive backoff. This effect is expected to differ with the initial window size and drop handling variant.

TCP Implementations: HTTP connection management schemes enable more and longer TCP connections outstanding at the same time at the Web server. More memory and computation resources could thus be taken up and for longer durations, with effect varying depending on the implementation and the TCP variant.

The HTTP and TCP parameters discussed above could interact with each other. However, it is not clear which parameters are more dominant and to what degree. To obtain a more rigorous understanding of the interaction between HTTP and TCP, we systematically study the impact of these parameters on a number of Web performance metrics such as time for first response, page download, and embedded object download, and server load. We seek to identify the parameters or combinations of parameters that matter the most and provide guidelines for a cost-efficient tuning of the Web interaction.

III. SIMULATION EXPERIMENTS

A. Simulation software

We use the ns-2 (Network Simulator version 2) [3] for our simulations. This discrete event simulator provides a rich library of modules including the different flavors of TCP and HTTP we investigate in this study. We made some changes in the simulator's code to fit it to our requirements, thus we extended the workload generation part to include the exact hybrid Web object size model in SURGE [4].

B. Simulation scenario

We use a simple dumbbell topology for our simulations. Two clouds of Web client and server nodes are on the edge of the topology. They are connected through a bottleneck link in the center. The edge links are configured to reasonably emulate the effective delay and bandwidth from

an end host to the backbone. The center link is configured to emulate the effective delay and bandwidth across the Internet backbone.

For comparison, we create one Web session between a Client and a Server nodes, vary the TCP and HTTP parameter values, and observe the performance changes. For Web session we use a combination of Pareto-based [5] and SURGE-based model for realistic inter-request time, number of embedded objects, and size of embedded object distributions.

To capture the effect of Web sessions interacting and interfering with each other, we generate a number of sessions in both directions. To incur different levels of traffic load or loss rate, we increase or decrease the amount of the cross traffic. Varying amounts of cross traffic drive loss rate into three different levels, approximately 0.1% (low-loss), 1% (medium-loss), and 3% (high-loss). We repeat the simulations with different random seeds to make sure that the loss rate remains in a reasonable range.

C. Parameter space

In Section II we discussed TCP and HTTP parameters of interest. For TCP parameters, we base our choice of the most common values and the realistic ranges from [6]. For HTTP parameters, we take those reported in [1]. For ease of comparison, we further define the base case being the one with the most commonly used parameter values (TCP variant: NewReno; MSS: 536 Byte; Delayed ACK: on; RTO: 3s; Timer granularity: 100ms; Initial Window Size: 2; Timestamp Option: on; HTTP connection style: serial-HTTP/1.1 (persistent); Nr. of parallel connections: 2).

D. Performance metrics

We examine four metrics: time for page download, first response, and object download, and server load. Page download time is difference between the time of issuing of a page request and the end of the page download. Object download time is the time difference between the request of an object and the end of the object download. First response time is defined as time between issuing of a page request and receiving of the index page header. By server load we mean the number of outstanding TCP connections on a server indicating the level of resources a server needs to allocate to keep these outstanding connections.

E. Results

We completed 30-mintue long simulations for all possible combinations. Each combination was repeated 10 times with different seeds. For the sake of simplicity, we compare cases of only one varying TCP parameter and fix other parameters to their base values. Additionally, we examine all possible combinations of HTTP parameters with TCP parameters set to the base values.

Page download time: Increasing maximum segment size clearly has positive impact while turning off delayed ACKs tends to have positive impact. We do not observe a clear trend of TCP variants improving or degrading Web performance. TCP SACK tends to be worst flavor in the high-loss case. Turning off persistency tends to have negative impact. Adding parallel connections helps in the low- and medium-loss cases.

First response time: Increasing segment size tends to have negative impact, but the effect is relatively smaller compared to the effect of disabling persistency. Other TCP parameters have very little influence on the first response time. Turning off persistent connections has clear negative impact. Pipelining does not improve the first response time. Increasing number of parallel connections does not reduce the first response time either.

Object download time: Increasing segment size has positive influence. Other TCP parameters do not have significant impact. Turning on pipelining however degrades overall object download time. Varying number of parallel connections has almost no effect on the object download time.

Server load: TCP parameters either do not have significant impact or do not have clear positive or negative impact. Turning on pipelining, turning off persistency, and reducing the number of parallel connections have clear positive impact.

IV. LIVE EXPERIMENT

Basically we have used extensive simulations to investigate the impact of various TCP flavors and control parameters on the performance of HTTP. To validate our simulation results we ran live measurements, as well.

A. Measurement Software

A key requirement of the measurement software is to be able to easily switch among different TCP flavors and to modify a range of TCP and HTTP options and parameters without any human intervention. Unfortunately, TCP is implemented as part of the kernel network stack in most modern operating systems. To bypass this problem we used a user-level network library called Alpine [7] in conjunction with a real Web server and a HTTP client.

The Alpine User-Level Networking Stack: Alpine is a user-level library that supports an unmodified FreeBSD networking stack on top of UNIX and can be configured directly at the user-level without having to recompile or reboot the kernel.

HTTP Client Software: We use `httperf` [8] as the basic engine for making all retrievals in our study. The `httperf` software is attractive because it allows a set of objects to be retrieved from a Web server using the variety of HTTP/1.0 and HTTP/1.1 protocol options of interest to our study.

HTTP Server Software: Our choice of the server software is limited by Alpine's inability to support `fork()`, which excludes popular servers like Apache from our candidate list. Thus, we used the `boa` [9] Web server in our study.

B. Test Environment

To avoid sending too much measurement traffic into the Internet, we decided to first explore the entire parameter space in a local environment. Our next goal is to use wide-area measurements to validate the major findings of our simulations and the local experiments. For the local environment we used `dummynet` [10] to simulate a range of delays and packet drops.

C. Experimental Results

For a given HTTP connection option and loss category we examined what combination of parameters are good from a performance viewpoint.

The best TCP flavor is Tahoe if the amount of data to be sent is limited and when the congestion window cannot grow too large. On the other end, SACK works well only when a large amount of data is to be sent on a connection. In the presence of loss and small congestion window, unnecessary retransmission can actually help since the flow of data and ACKs is maintained. However, since SACK generally attempts to lower unnecessary retransmissions, we found that it tended to perform not as well as other flavors in the scenarios we tested. Improving MSS provided the best performance. A good value for initial window size is 4 in most cases. Turning off delayed ACKs is clearly an improvement. The RTO and timer granularity parameter values are generally of less consequence. The timestamp option is generally useful.

V. Analysis and Conclusion

A. Impact of TCP Parameters

The results regarding impact of TCP parameters on performance of Web servers are generally in line with our expectations. We found that increasing MSS, and turning off delayed ACK result in better overall performance. Other TCP parameters do not have any significant impact on the performance. One interesting, if not unexpected, result was that the particular variant of TCP (i.e., Reno, NewReno, etc.) in use does not have much impact on the overall performance of the Web server. This is a surprising result. It is indeed known that Tahoe offers performance benefits over Reno in some cases [11]. Similarly, use of selective acknowledgment (SACK) option did little to improve performance. Since we are only talking about average performance, it is still possible that under a specific loss pattern, individual HTTP transfers may see performance benefits from using options like SACK.

B. Impact of HTTP Parameters

We examined different connection methodologies in both HTTP/1.0 and HTTP/1.1. We examined the use of 1 or more serial HTTP/1.0 connections, serial HTTP/1.1 connections, as well as pipelining requests over the persistent connections. We ratified the fact that pipelining over persistent connections offers the best performance and serial HTTP/1.0 offers the worst performance. The fact that serial HTTP/1.0 is the worst and burst HTTP/1.1 is the best is not entirely novel [1] but it is still useful to verify the full ordering via simulation and live experiments.

C. Combined Impact of TCP and HTTP Parameters

The overall impact of TCP is much smaller than HTTP and thus improving the HTTP connection mechanisms can provide better Web performance. However, there are several subtle interactions between TCP and HTTP as we have seen in our experiments. Under high loss rate, for flows of short or medium durations created by serial HTTP/1.1 connections, trying to save a few unnecessary retransmissions can worsen the performance. For short transfers, all variants of TCP under high loss behave similarly. But for larger transfers, SACK may prove to be more useful. With serial HTTP/1.1, we see some application-limited thin and long flows. Here, "unnecessary" retransmissions turn out to be beneficial, making Tahoe often to be the best performer, and SACK not so well.

Summarizing our investigations we can say that some of our results confirm the common settings in modern Web servers while others indicate that closer attention needs to be paid to some parameters. Tuning HTTP parameters has much more impact than tuning TCP parameters. Our further goals are to validate the major findings of our simulations and the local experiments in a wide area and to see if examination of other parameters or combinations might be warranted.

VI. References

[1] B. Krishnamurthy, C. E. Wills, "Analyzing Factors that influence end-to-end Web performance", In *Proc. World Wide Web Conference*, pages 17–32, May 2000.

[2] K. Fall, S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", *Computer Communication Review*, 26(3), Jul. 1996.

[3] L. Breslau et al., "Advances in Network Simulation", *IEEE Computer*, May 2000.

[4] P. Barford, M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", In *Proc. ACM SIGMETRICS*, June 1998.

[5] A. Feldmann, A. C. Gilbert, P. Huang, W.Willinger, "Dynamics of IP traffic: A study of the role of variability and the impact of control", In *Proceedings of the ACM SIGCOMM*, Cambridge, MA, August 1999.

[6] J. Padhye, S. Floyd, "On Inferring TCP Behavior", In *Proc. ACM SIGCOMM '2001*, Aug. 2001.

[7] D. Ely, S. Savage, D. Wetheral, "Alpine: A User-Level Infrastructure for Network Protocol Development", In *Proc. USITS '01*, Mar. 2001.

[8] D. Mosberger, T. Jin, "`httperf` - A Tool for Measuring Web Server Performance", In *Workshop on Internet Server Performance*, June 1998.

[9] L. Doolittle, J. Nelson, "The BOA Web server", available at `http://www.boa.org/`

[10] L. Rizzo, "Dummynet: A Simple Approach to the Evaluation of Network Protocols", *Computer Communication Review*, 27(2), Feb. 1997.

[11] A. Kumar, "Comparative Performance Analysis of Versions of TCP in local network with a lossy link", *IEEE/ACM Transactions on Networking*, 6(4), Aug 1998.