CableLabs®

ACCESS NETWORK TECHNOLOGIES

ANALYSIS OF GOOGLE SPDY AND TCP INITCWND

prepared by **Greg White**
Principal Architect
Systems Architecture & Design
g.white@cablelabs.com

CableLabs Program/Project Leads:

Jean-François Mulé, Technology Development
jf.mule@cablelabs.com

Dan Rice, Access Network Technologies Program Lead
d.rice@cablelabs.com

Rev. 3
May, 2012

DISCLAIMER

ACKNOWLEDGMENTS

The author would like to thank Marc Weaver, Darshak Thakore, Eric Winkelman, Robin Sam Ku, Scott Maize, and Rob Moon for their assistance in developing and configuring the test bed, and collecting the data presented in this report. The author would also like to gratefully express his appreciation to Ken Barringer and Christie Poland for manuscript preparation assistance.

Table of Contents

List of Figures

List of Tables

EXECUTIVE SUMMARY

Making the Internet faster is the goal of many product and service companies. Many strategies exist, from data caching to packet switching, performance improvements in Internet browser clients and server software, without forgetting the transport network scaling from the fiber core to the access edges.

This report investigates two proposed protocol enhancements aimed at making the *web browsing* user experience better by optimizing the transport of web objects and thereby reducing the page load time. The two enhancements are Google SPDY, a replacement of the HyperText Transfer Protocol (HTTP) requiring client and server changes, and a TCP tune-up, an increase in the TCP initial congestion window accomplished by a setting change on the web servers. Both show promise, but there are some significant caveats, particularly with SPDY. SPDY is considered to be a strong candidate for standardization as HTTP/2.0.

In addition to a discussion of the two enhancements, this report provides the results of laboratory testing on SPDY version 2 and the proposed increase in the TCP initial congestion window in a variety of simulated conditions, comparing the page load time when using the proposed enhancement to the page load time for the default case of HTTPS and current initial congestion window settings.

The proposed enhancements generate mixed results: web page load times were reduced in some scenarios but increased significantly in others. The performance improvement (or degradation) varied depending on the number of servers, configuration of the initial TCP congestion window, and especially any network packet loss. The following results were obtained across all scenarios comparing SPDY and congestion window enhancements to standard HTTPS.

- Average reduction in page load time was 29%
- Best improvement was over 78% reduction in page load time
- Worst cases showed a negative impact, resulting in a 3.3x increase in page load time

These results lead us to the following conclusions:

- The SPDY protocol is currently a moving target, and thus it would be challenging to realize a return on investment for general-purpose usage in MSO servers.
- Protocol improvements, standardization in the IETF and wider adoption by the client/server software may warrant a second look at SPDY.
- **Some applications in controlled environments may gain by leveraging SPDY.** SPDY might be a valuable tool where the MSO provides the servers, the client software, and the web content. An example application might be an HTTP-delivered remote user interface delivered to a Set-Top Box or IP video client.
- If SPDY were adopted very widely it may have some secondary benefits for network operators through improved infrastructure scalability due to a significant reduction in concurrent TCP sessions, as well as a reduction in Packets Per Second.
- **The proposed increase in the TCP initial congestion window is straightforward,** requires no client modifications, and on its own provides consistent (albeit modest) performance gains.

1 INTRODUCTION

The current method of transporting web objects from a server to a client utilizes Hypertext Transfer Protocol version 1.1 (HTTP/1.1), running atop the Transport Control Protocol (TCP). HTTP/1.1 was published as an IETF RFC in 1999, and has since become the most widely used application-layer protocol on the Internet. TCP pre-dates HTTP/1.1 by about 18 years, and even though TCP has evolved over time, it is fundamentally unchanged, and runs atop IPv4 and IPv6.

Since the advent of the World Wide Web 15+ years ago, access network bandwidths as well as server and client CPU horsepower have increased by a factor of approximately 1.5x year-over-year, yet the apparent speed of the web (from the web browsing user's perspective) has grown much more slowly. In part, this can be explained by a concomitant increase in web page complexity, as measured by any number of attributes including total page size, number of linked resources, amount of server-side code, and lines of JavaScript. However, there is a view that the underlying protocols used to transport web resources are becoming increasingly out-of-date with the network and computing resources that are in use today, and that significant improvements in performance (generally page load time) can be achieved by revisiting these fundamental technologies.

The user perception of the speed of the Internet–the time it takes to load a web page–has not followed Moore's law.

Bolstering this view is the fact that while network bandwidths have been improving rapidly, network latencies have not, and there is little hope of achieving significant reductions in network latency, as it is dominated by propagation delay. The result is that a fundamental network parameter, the Bandwidth-Delay Product (BDP), is much larger today than in the early days of the Internet. It is becoming clear that HTTP/1.1 and TCP are not particularly optimized for networks with high bandwidth-delay product. Figure 1 shows Google's view of the diminishing returns for HTTP/1.1 as network bandwidth increases.

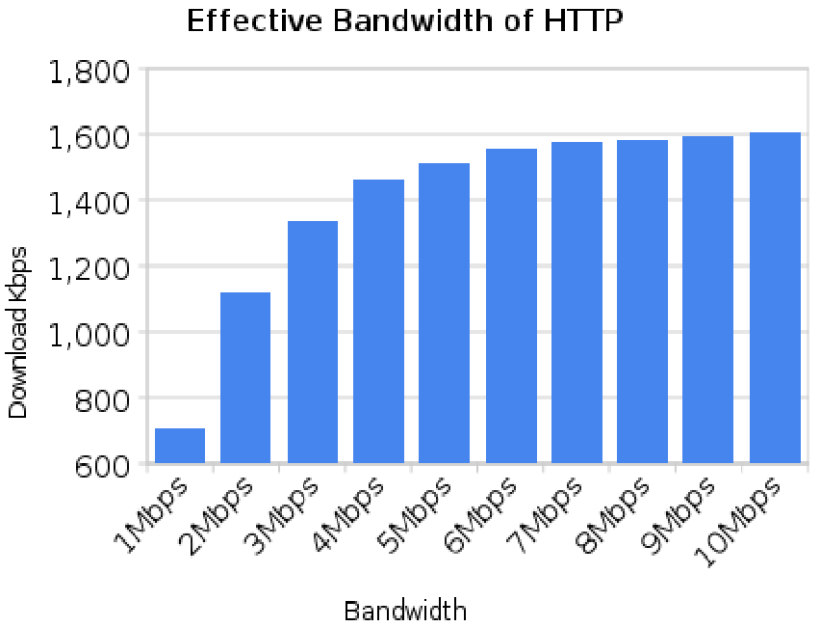

Figure 1. Effective Bandwidth of HTTP

[Source: SPDY Essentials, Roberto Peon & William Chan, Google Tech Talk, 12/8/11]

Furthermore, in order to maximize performance using the existing tools of HTTP/1.1 & TCP, web architects (browser, server and site developers) have employed work-arounds that have some negative implications. The most significant implications result from the use of multiple simultaneous TCP connections. Modern browsers will open up to six simultaneous TCP connections to each server from which they need to retrieve resources, and optimized websites will compound this by spreading content over multiple servers, a practice known as "domain sharding". The result is that a single browser may have 20 or more simultaneous TCP connections to the hosts of a particular site while it is downloading a single web page. The parallel nature of this approach is intended to reduce the page download time (compared to the alternative of a single, non-pipelined TCP connection), and in many cases it succeeds. However, a web browser with 20 or more simultaneous TCP sessions can drown out other applications that use a small number of TCP sessions (e.g., one). This is due to the fact that the TCP congestion avoidance algorithm provides approximately fair sharing of the bandwidth on the bottleneck link on a per-TCP-session basis. Additionally, some browsers do not reuse TCP connections, so each web resource is retrieved using a separate TCP connection. The result is that the vast majority of sessions never reach the congestion avoidance phase, and network resources can therefore either be underutilized or excessively congested. The network overhead for establishing and tearing down all of the TCP connections can also be a concern.

A number of efforts have sprung up to develop ways to improve page download time and consequently improve the web browsing user experience. This paper discusses two proposed enhancements, both originally proposed by Google as part of their "Let's Make the Web Faster" project [http://code.google.com/speed/]. The first is a replacement for HTTP/1.1, called SPDY, which aims to make more efficient use of the network in common web browsing scenarios. The second is an incremental enhancement to the TCP protocol in the form of an increase in the server's initial congestion window (initcwnd).

In addition to a discussion of the two proposed enhancements, this paper presents the results of laboratory testing conducted by CableLabs on both technologies independently, and on the combination of the two technologies.

Could TCP or HTTP optimization help improve the load page time?

Two proposed solutions are tested in this report.

2 SPDY

SPDY [http://www.chromium.org/spdy] is an experimental protocol developed by Mike Belshe and Roberto Peon of Google in late 2009 to replace HTTP/1.1 communication for transporting web content between a client and a server. It is deployed in production on many Google servers but requires a compatible browser such as Google Chrome. The stated goals of SPDY are:

- 50% reduction in page load time
- Minimize deployment complexity
- Avoid changes to content by web authors
- Solve this collaboratively via open-source software

At the time of this study, the current implemented version of SPDY is version 2 (SPDY/2). Version 3 is currently being developed.

2.1 The SPDY Protocol

SPDY/2 retains the HTTP/1.1 metadata, but replaces the transport aspects of HTTP with a more streamlined approach. SPDY has three basic features that are used to accelerate the loading of a page:

- **Multiplexed streams** – The fundamental enhancement of SPDY is that multiple resources can be retrieved via a single TCP connection. The expectation (and current implementation in Chrome) is for the client to open a single TCP connection to each server, and to request all resources of the server over that single connection. The use of a single TCP connection in this way allows the congestion avoidance algorithm in TCP to more effectively manage data flow across the network. Also, the client can include multiple requests in a single message, thereby reducing overhead and the number of round-trip times necessary to begin file transfer for requests beyond the first. While this is similar to HTTP pipelining [http://en.wikipedia.org/wiki/HTTP_pipelining], the difference introduced by SPDY is that the server can transfer all of the resources in parallel (multiplexed) without the "head-of-line blocking" problem that can occur with HTTP pipelining.
- **Request prioritization** – While SPDY may serve to decrease total page load time, one side-effect of stream multiplexing is that a critical page resource might be delivered more slowly due to the concurrent delivery of a less critical resource. To prevent this, SPDY provides a way for the client to indicate relative priorities for the requested resources. For example, if a JavaScript library is required in order to generate URLs for additional page resources, the client can request that the library be delivered with higher priority so that it isn't holding up those subsequent requests.
- **HTTP header compression** – SPDY mandates that HTTP headers be compressed to reduce the number of bytes transferred. HTTP Header compression has been an Internet standard but it is not widely used. Google SPDY makes it mandatory. This might be a fairly small gain on the response headers, but it can provide a significant benefit on the requests (which in many cases are entirely headers). For example, each request from a particular client to a particular server includes an identical user-agent string (which can be in excess of 100 bytes), and in some cases the same cookie is sent in each request. Both Chromium and Firefox developers have reported approximately 90% header compression using the proposed zlib compression. This could allow a lot more requests to be packed into each request packet.

SPDY also has two advanced features that can be used in certain cases to further accelerate the web user experience:

- **Server Push** – SPDY allows the server to create a stream to the client, and via this stream send web resources that were not explicitly requested by the client. The expectation is that the client will cache the pushed resource, and then upon needing it, will retrieve it from local cache. This function could potentially be used by the server to push objects (of which the client isn't yet aware) that are necessary for rendering the current page. Additionally, this function could be used to push resources for related pages that the user may be likely to request. The heuristics used by the server to decide when/if to push objects and what those objects are is left to the server implementer. This feature is somewhat controversial, but the authors defend it by pointing out that it is better than the practice of "in-lining" resources into the html page, since it allows the pushed resources to be cached for multiple future usages.
- **Server Hint** – SPDY defines a new header that the server can use to suggest additional resources that the client should request. This is a less forceful approach than the Server Push, and allows the client to be involved in the decision whether or not a particular resource is delivered. The client might, for example, examine its own cache and only request resources that are not resident in local cache. On the other hand, Server Hint theoretically requires one additional round trip that would be eliminated by Server Push.

SPDY/2 runs solely over an encrypted (TLS) connection. The rationale for this is three-fold:

1. TLS involves a client-server handshake to negotiate cipher-suite. The authors of SPDY extend this handshake to negotiate whether SPDY can be used. This Next Protocol Negotiation (NPN) [http://tools.ietf.org/id/draft-agl-tls-nextprotoneg] allows the use of the https:// URI, rather than a new spdy:// URI, and as a result a single html page can work for clients that support SPDY and clients that don't.
2. TLS passes through firewalls and bypasses intermediaries. Many HTTP requests today are processed (and modified) by transparent proxies without the knowledge of the end-user. It would create a tremendous barrier to adoption of SPDY if it were necessary for intermediaries to be upgraded to handle the new protocol.
3. Encryption is good. The authors of SPDY state a philosophical belief that all HTTP traffic should be encrypted. They cite the relative ease by which traffic (particularly Wi-Fi traffic) can be snooped.

2.2 SPDY SERVER IMPLEMENTATIONS

There are a number of SPDY server implementations that are in various stages of development as open source projects. The Chromium project maintains a list of the implementations available [http://www.chromium.org/spdy/]. Many of these implementations currently support a subset of SPDY functionality and/or support SPDY version 1. At the time of drafting this report, only two implementations appear to support SPDY/2 in a reliable way. The first is the Chromium "FLIP Server" (FLIP was an early internal code-name for SPDY within Google). This server is built off of the immense Chromium source tree. The second is a Javascript implementation called "node-spdy" which is built on the node.js server framework.

Work was started on an Apache module for SPDY, but it is currently incomplete.

Google SPDY is deployed by Google for gmail, Picasa, Google+, Google Docs and Amazon for its Kindle Fire Silk web browser

2.3 SPDY Client Implementations

On the client side, the Chrome browser is the most important current implementation. The current version of Chrome supports SPDY/2 by default, and includes a built-in diagnostics function that allows the user to examine the active SPDY sessions [chrome://net-internals/#spdy]. The Mozilla Firefox browser includes SPDY/2 support as well, but is only available in the Firefox 11 "aurora" builds (the current mainline version is Firefox 9), and is disabled by default. Finally, the Silk browser in the Amazon Kindle Fire tablet computer purportedly utilizes SPDY/2 for its connection to the Amazon EC2 cloud when performing web acceleration.

2.4 SPDY in the Wild

In terms of live web content, the most significant, publicly available sites that serve content via SPDY are run by Google. Many of the Google properties have SPDY/2 enabled, including Gmail, Google Docs, Picasa, Google+, and Google Encrypted Search. All of these sites utilize only the basic SPDY features; there are no known live instances of Server Push or Server Hint. In addition, the web acceleration companies Cotendo (acquired by Akamai in Dec. 2011) and Strangeloop indicate that they have deployed SPDY in some capacity.

2.5 SPDY Protocol Development and IETF Standardization

Google has actively solicited input on enhancements to the SPDY/2 protocol. Up until very recently, development and discussion of SPDY/3 has taken place on an open, Google-managed forum. However, the SPDY/3 draft was submitted to the IETF HTTPbis working group on February 23, 2012, for comments [http://tools.ietf.org/id/draft-mbelshe-httpbis-spdy], and it is expected that some if not all of the further development will take place on the HTTPbis mailing list. Along with the draft came an IPR declaration by Google, which provides a royalty-free-with-reciprocity license.

In addition, on January 24, 2012, the chair of the IETF HTTPbis WG proposed that work begin on an HTTP/2.0 based at least in part on SPDY. This proposal has sparked intense discussion in the HTTPbis working group. While it appears that many support the initiation of work on HTTP/2.0, it is less clear how much of SPDY will end up being an integral part of it.

3 TCP INITIAL CONGESTION WINDOW

The TCP initial congestion window (initcwnd) is used at the start of a TCP connection. In the context of an HTTP session, the server's initcwnd setting controls how many data packets will be sent in the first burst of data from the server. It is a standard protocol parameter that can be changed on Linux servers via a simple command line.

Absent packet loss or receiver window limits, the TCP slow start operation looks like:

Table 1. TCP Slow Start Operation

Round-Trip #	Client	Server
1	TCP SYN	TCP SYN/ACK
2	TCP ACK & HTTP GET <*url*>	initcwnd data packets
3	TCP ACKs	2*initcwnd data packets
4	TCP ACKs	4*initcwnd data packets
		and so on....

3.1 HISTORICAL SETTINGS FOR INITCWND

A larger value for initcwnd will clearly result in fewer Round-Trip Times (RTTs) to deliver a file. However, the downside to an excessively large initcwnd is that there is an increased risk of overflowing a router buffer on an intermediate hop, resulting in an increase in latency from packet loss and retransmissions. A value of initcwnd that is greater than the Bandwidth-Delay Product (BDP) of the network path between the server and client has an increased likelihood of causing packet loss that may lead to poor performance.

As network bandwidths have increased over time, the BDP has increased, and as a result the defined value for initcwnd has been adjusted. The early specifications for TCP [RFC1122, RFC2001] required that a TCP implementation set initcwnd to 1 packet. Starting in 2002 [RFC2414, RFC3390], the initcwnd value was raised to 4000 bytes (effectively 3 packets in most networks).

Google has submitted an IETF draft [http://tools.ietf.org/id/draft-ietf-tcpm-initcwnd] proposing that initcwnd now be increased to at least 10 packets, based on a series of tests performed using production Google servers [http://code.google.com/speed/articles/tcp_initcwnd_paper.pdf]. In Google's tests, a value of 10-16 packets resulted in the minimum average latency for delivery of web resources.

In November 2011, CDN Planet [http://www.cdnplanet.com] performed an assessment of CDNs to see what value is currently used for initcwnd. Many CDNs have already increased their initcwnd beyond 3 packets, some as high as 10-16 packets.

initcwind is the number of TCP packets sent during the initial burst of data. By default, web servers use an initcwnd setting of 3.

The proposed enhancement is to set the default setting to 10.

Some CDNs have increased it to 10 and even 16.
Linux 2.6.39 increased the default window to 10.

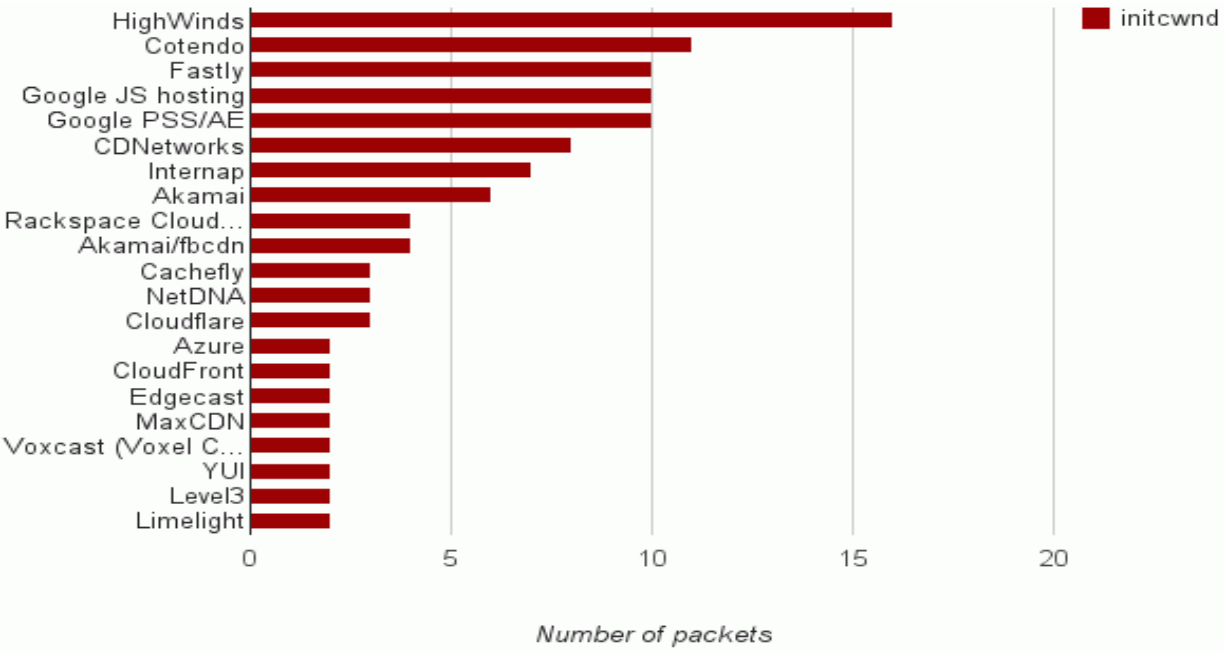

Figure 2. Initcwnd Values in Use by CDNs

[Source: www.cdnplanet.com Nov 16, 2011]

This increase from a value of 3 to a value of 10 will result in the elimination of up to 2 RTTs for each file transfer. The maximum gain will be seen for files that take 9 packets to deliver (i.e., files around 11 kB in size). Files of that size would take 4 RTTs to deliver using the default initcwnd, but can be delivered in 2 RTTs with the proposed increase, a 50% improvement. Files smaller than about 3 kB will not experience any acceleration. Figure 3 shows the expected percent reduction in file download time as a function of file size under the following assumptions:

- 100ms RTT
- 10 Mbps Downstream limit
- 0% Random Packet Loss
- TCP Receiver Window of 512 KB

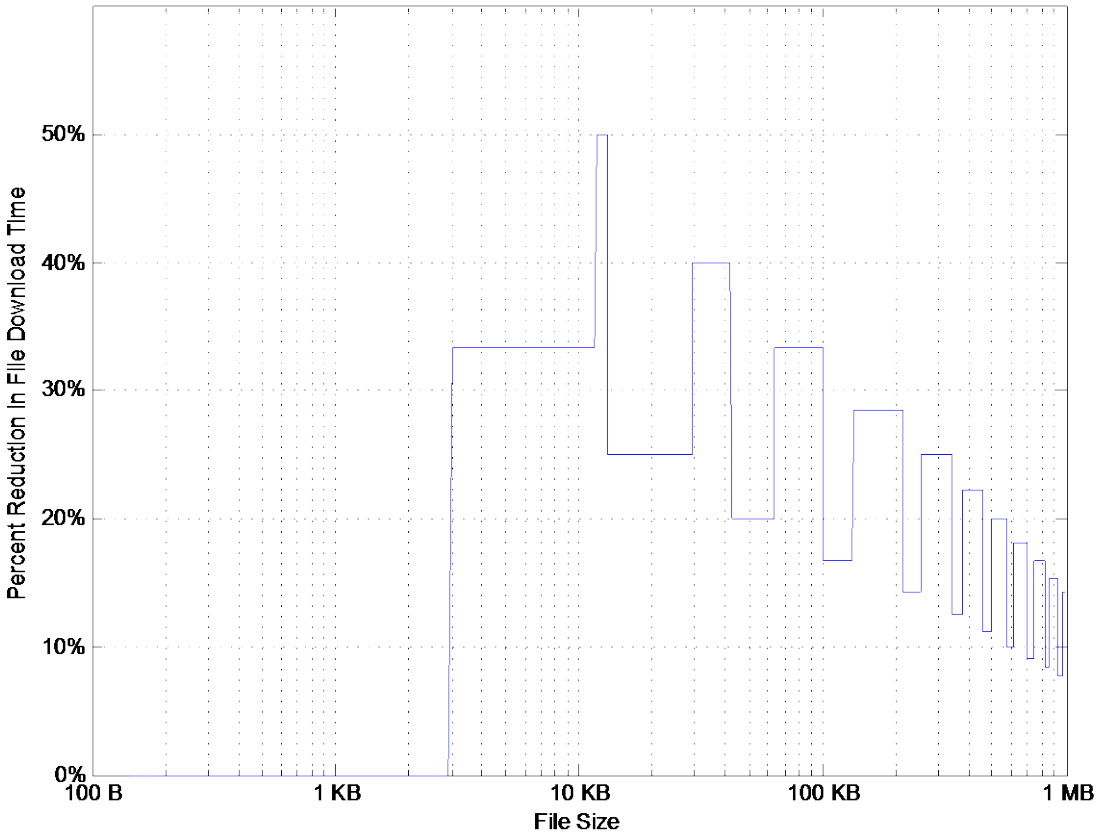

Figure 3. Expected Benefit in File Download Time Resulting from Initcwnd Increase

4 THE CURRENT WEB ENVIRONMENT

4.1 CONTENT ENVIRONMENT

The construction of web page content plays a very important role in determining the page load time. In order to validate the two proposed enhancements, we will create a test bed that consists of web servers, web clients and web content. We performed a survey of some popular (and resource heavy) websites in order to understand the content environment. The results shown in Table 2 indicate, for each site's home page, the number of servers from which content was drawn, the number of resources requested, the total transferred size of all page resources, and the page load time. The reported page load time is the total time to download all of the resources for the page. It is possible, and even likely, that one or more of these pages are largely complete (perhaps only missing some non-critical resources) in significantly less time than is reported here.

Additionally, for each webpage we evaluated the percentage of total resources that were requested from the top N servers for the page, for values of N between 1 and 5. For example, Engadget receives 34% of the web objects from the most used server, 82% of the objects from 3 most used servers and 87% of them come from the 5 servers with the highest count.

These servers may include the host servers for the website, CDN servers for images and media content, advertising servers for ad content, and analytics collection, among other types of resources.

Table 2. Website Survey

					% of GETs from top N servers				
Page	**servers**	**GETs**	**total page size (KB)**	**total time (s)**	**N=1**	**N=2**	**N=3**	**N=4**	**N=5**
Engadget	26	278	1500	23	34%	67%	82%	85%	87%
NY Times	27	148	1500	13.6	33%	49%	59%	67%	73%
Gizmodo	25	116	3900	13.2	34%	60%	69%	76%	78%
CNN	22	158	1180	12.6	59%	70%	75%	80%	83%
comcast.net	12	55	606	12	51%	67%	75%	80%	84%
ESPN	26	144	5400	11.6	40%	51%	60%	67%	70%
Amazon	15	139	1100	11	34%	58%	81%	86%	90%
RoadRunner	30	128	85	9.5	27%	48%	61%	65%	68%
Wired	32	130	1200	8.9	48%	56%	61%	65%	68%
eBay	14	53	520	8.6	23%	40%	55%	68%	75%
LinkedIn	9	75	457	7.17	48%	80%	88%	92%	95%

					% of GETs from top N servers				
Page	**servers**	**GETs**	**total page size (KB)**	**total time (s)**	**N=1**	**N=2**	**N=3**	**N=4**	**N=5**
Hulu	12	193	2000	4.96	60%	79%	87%	95%	96%
Yahoo	6	51	423	4.6	88%	92%	94%	96%	98%
YouTube	6	37	2400	4.16	41%	68%	81%	92%	97%
Google shopping	3	20	73	2.45	45%	90%	100%	100%	100%
Average	**18**	**115**	**1490**	**9.8**	**44%**	**65%**	**75%**	**81%**	**84%**

Average website:
115 web objects totaling 1.5 MB from 18 different servers
80% of content comes from the top 4 servers

4.2 Network Environment

The network environment between the servers and the client plays an equally large role in determining page load time. The most important parameters that drive performance are the round-trip time, the bottleneck link bandwidth, and the packet loss rate.

4.2.1 Round-Trip Time (RTT)

In the absence of congestion, typical Internet round-trip times can range from as low as 10 milliseconds when accessing content served in close proximity, to as high as 500 milliseconds or more when accessing servers across continents. For wireline broadband customers in the U.S., the most popular websites can generally be reached with RTTs ranging from 15 ms to 150 ms. For the websites included in the survey in Section 4.1, a sample TCP RTT to each of the top-five servers (in terms of number of resources requested) from CableLabs headquarters in Louisville, Colorado, is shown (in milliseconds) in Table 3 along with a summary of the minimum, mean, and max RTT of those top-five servers. The RTT was calculated by doing a TCP "ACK RTT" analysis of a packet capture of each page load using the Wireshark tool. The ISP connection used during this testing consisted of two load-balanced 100 Mbps duplex links with a mean RTT of less than 2 ms.

Table 3. Round-Trip Times to Website Servers in Milliseconds

Page	server 1	server 2	server 3	server 4	server 5	min	average	max
Engadget	63*	62*	64*	62*	38	38	57.8	64
NY Times	64*	64*	62*	64*	64*	62	63.6	64
Gizmodo	37*	39*	**134**	25	22	22	51.4	134
CNN	46*	53	69*	32	45	32	49.0	69
Comcast.net	65*	69*	94	28	71	28	65.4	94
ESPN	64*	33	65*	64*	64*	33	58.0	65
Amazon	35	72*	**16***	98	33	16	50.8	98
RoadRunner	56	58	29*	52*	61*	29	51.2	61
Wired	70*	114	66*	24*	85	24	71.8	114
eBay	62*	75	64*	62*	62	62	65.0	75
LinkedIn	50*	72*	78	54	63*	50	63.4	78
Hulu	65*	67*	66*	64*	22	22	56.8	67
Yahoo	28	27	64	29	61	27	41.8	64
YouTube	23	25	23	22	22	22	23.0	25
Google shopping	34	42	43	22	-	22	35.3	43

* denotes a host run by a CDN provider.

The RTT measurements on sample sites
Minimum RTT: 16ms
Maximum RTT: 134ms
Mean RTT: 53.8 ms

When network links are congested, a phenomenon referred to as "bufferbloat" can result in an increase in RTT on the order of hundreds of milliseconds. Bufferbloat arises due to the fact that many network elements support much more buffering than is needed to maintain full link utilization. These oversized buffers will be kept full by the TCP (particularly when there are multiple simultaneous sessions), thereby

causing an increase in latency. In recent updates to the DOCSIS 3.0 specification, a new feature has been added to mitigate this effect by proper sizing of the cable modem's upstream buffer. At this time, 14 cable modem models from 10 manufacturers have been certified as supporting this feature. For more detail on bufferbloat and the DOCSIS Buffer Control feature, see: [http://www.cablelabs.com/specifications/CM-GL-Buffer-V01-110915.pdf].

4.2.2 Access Network Data Rate

Residential access network data rates have steadily risen over time, with a 10 Mbps downstream rate and 1 Mbps upstream rate being fairly common in North America, and 20 Mbps x 2 Mbps rates becoming increasingly available. Many operators offer even higher tiers, with DOCSIS speeds in Europe up to 360 Mbps on the very high end, but typically they currently have fairly low take rates. Some access networks may have lower speed limitations placed on the services such as public and outdoor Wi-Fi currently being deployed by MSOs. For purposes of our investigation of the two proposed technologies, we will utilize the 10x1 and 20x2 data rate configurations.

4.2.3 Packet Loss

It is important to note that packet loss due to router (or switch) buffer overflow is expected behavior in networks, particularly when TCP is utilized. In fact, packet loss due to buffer overflow is fundamental to the congestion avoidance algorithm in TCP; it is the only signal that the TCP has (absent the Explicit Congestion Notification field in IP packets) that it has saturated the link and needs to reduce its transmission rate. As a result, it is not a concern in this testing. The concern here is random packet loss due to noise, interference or network faults that will effectively send TCP an erroneous signal to reduce its transmission rate.

In wired networks, packet loss due to noise is fairly uncommon, with packet loss rates of approximately 10^{-5} being typical for DOCSIS networks. In wireless networks, packet loss can vary dramatically as a result of interference and fluctuations in carrier-to-noise ratio.

5 LABORATORY TESTING AND RESULTS

5.1 GOALS

The goals of the testing are to investigate the page load performance impacts of the two proposed technologies. In particular, we examine Google SPDY with an eye toward evaluating SPDY's ability to achieve the stated performance goal and to understand what aspects of SPDY provide acceleration and in which scenarios. Testing is limited to the SPDY "basic" features; no testing of server-push or server-hint is included in this study, primarily due to the fact that the performance enhancement gained by using them would be heavily dependent on how they are configured and utilized by the website administrator. Some benchmarks of server-push and server-hint have been published by external sources; see [https://docs.google.com/View?id=d446246_0cc6c6dkr]. The results have not shown a statistically significant gain.

As stated previously, the TCP initcwnd increase is tested both independently (using HTTPS) and in conjunction with SPDY.

5.2 LABORATORY ENVIRONMENT

The entire test network was virtualized on a dedicated VMWare ESX Virtual Server. The test network consisted of 7 virtual hosts, each having two network interfaces. A single network interface on each host was connected to a virtual LAN within the ESX environment. The second network interface on each host was connected to the corporate network. During the experiments, the CPU load on the virtualization server was checked to ensure that it was not impacting the results.

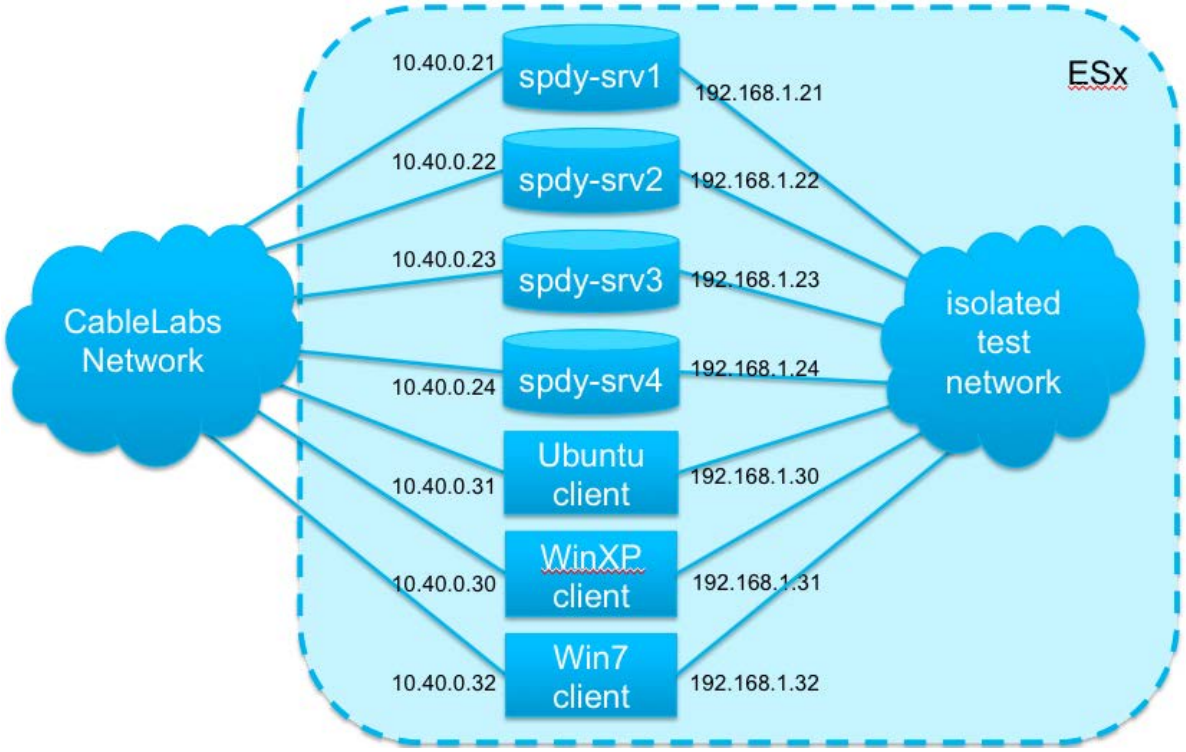

Figure 4. Test Bed Configuration

5.2.1 Web Server Configuration

Four of the hosts were configured identically as web servers, with the following software:

- OS: Ubuntu 11.04
- Web Server: “Express-SPDY” written in JavaScript for node.js

The "Express-SPDY" server is the melding of a compact and efficient open-source HTTPS server (known as "Express") with an open-source SPDY/2 implementation called "node-spdy". The implementation is written in JavaScript using the node.js server environment. This implementation was chosen over the Chromium package published by Google due to the fact that it was lightweight and could be deployed relatively easily. Unfortunately, the version of Express-SPDY available at the time of this testing was found to not be stable, and required some work to debug and fix prior to beginning any testing. Since our testing commenced, the author of node-spdy rewrote a large part of the package.

5.2.2 Client Configurations

The remaining three hosts were configured as client machines. The primary client ran Windows 7, and the other two clients (used for spot testing) ran Windows XP and Ubuntu 11.04.

All clients used the Chrome 16 browser, with the SPDY Benchmarking plugin, and were installed with Wireshark to capture network traffic.

The Dummynet [http://info.iet.unipi.it/~luigi/dummynet/] network simulator was used to simulate the various network conditions.

5.3 Test Conditions

288 Test Variants executed 20-100 times each.
3 protocol enhancement options tested
3 typical web pages served across varying number of web servers

5.3.1 Protocol Options

As previously stated, the testing is intended to compare the SPDY protocol to the HTTPS protocol, and to investigate the use of the increased TCP Initial Congestion Window (initcwnd) (both with HTTPS and in conjunction with SPDY). As a result, we measured the time it took to load a web page using four different protocol options as shown in Table 4. One option defines the baseline scenario or “Base Case”: with HTTPS for transport and the default initcwnd setting of 3. The results for the three cases involving a proposed enhancement are then compared to the "Base Case".

Table 4. Matrix of Protocol Options

	HTTPS	SPDY
initcwnd = 3	Base Case	CASE 1
initcwnd = 10	CASE 2	CASE 3

5.3.2 Web Sites

Nine different "websites" were used to evaluate the impact that the content and server configuration would have on the test results. The nine websites were formed by the 3x3 matrix of three different server configurations and three different web pages as follows.

5.3.2.1 Server Configurations

The server configuration parameter determined how all of the web resources for a page were served.

1. Server Case 1, all resources were served by a single server.
2. Server Case 2, the resources were divided equally across two servers.
3. Server Case 3, the resources were divided equally across four servers.

5.3.2.2 Web Pages

Three web pages were used in testing. These pages were chosen to span a range of possible page configurations.

3 sample web pages used
Page B most closely models a typical page
based on the sites surveyed in Section 4.1.

Page A consisted of 102 total resources, with a total page size of 369 KB. It was composed of the following resources:

- 1 HTML file: 12.8 KB
- 1 JavaScript Library (jquery.js): 94.5 KB
- 100 JPEG images: min/mean/max/stddev size = 1.3/2.6/5.5/1.2 KB

Page B consisted of 101 total resources, with a total page size of 1.4 MB. It was composed of the following resources:

- 1 HTML file: 8.2 KB
- 100 JPEG/PNG/GIF images:
  - min/mean/max/stddev size = 46B/14KB/103KB/24KB
  - Approximately log-uniform size distribution

Page C consisted of 11 total resources, with a total page size of 3.0 MB. It was composed of the following resources:

- 1 HTML file: 0.8 KB
- 10 JPEG images: min/mean/max/stddev size = 298/302/310/4 KB

5.3.3 Channel Conditions

Eight different channel conditions were used in the testing. These eight cases were formed from the 2x2x2 matrix of the following parameters:

- Configured Data Rate:
  - 10 Mbps downstream, 1 Mbps upstream
  - 20 Mbps downstream, 2 Mbps upstream
- Round Trip Time:
  - 20 ms
  - 100 ms
- Packet Loss Rate:
  - 0%
  - 1%

The two Configured Data Rates were chosen to match two commonly used residential high-speed data service configurations. The Round Trip Times correspond to a fairly short RTT that models a user accessing a site that is located fairly close to them (or is accelerated by a CDN), as well as a longer RTT which models a U.S. coast-to-coast page load or a case where there is increased upstream latency due to bufferbloat. The two packet loss rates correspond to an idealistic case that might be approached by an end-to-end wired network connection and a case with packet loss that might be more typical of a network connection that includes a Wi-Fi link.

5.3.4 Test Execution

The test execution will be performed in the following fashion:

- Run all 288 test conditions on Win 7
- Run spot tests on WinXP and Ubuntu
- For each test condition, perform 20 page loads (clearing cache and resetting connections before each load) and calculate median load time.

5.4 Test Results

The result of the spot tests on WinXP and Ubuntu were consistent with the results for the Win7 testing. The raw data for the WinXP and Ubuntu test cases are provided in the appendix, but will not be discussed further in this report.

As described in Section 5.3.1, the 2x2 matrix of protocol options will be presented as three different "cases", where each case represents the gain achieved by using one of the three proposed protocol enhancement options compared to the base case. This section is broken into three subsections that correspond to the three cases.

For each case, the results comprise a five-dimensional matrix of test conditions. In order to present the results in a compact way, each subsection below will first examine the impact that website configuration has on the achieved gain, then second will examine the impact of the channel conditions.

The results provide a comparison between the median web page download time achieved using the proposed protocol enhancement option and that which is achieved in the base case.

5.4.1 Case 1: SPDY vs. HTTPS

Case 1 compares the median web page download time achieved using SPDY/2 to the median time achieved using HTTPS. The comparison is made for each website configuration and channel condition combination. The comparison is reported as a "gain", which is calculated as the ratio of median page load time using HTTPS to the median page load time using SPDY. As a result, gain values greater than 1 indicate that SPDY/2 provides an advantage over HTTPS. As an additional point of reference, a gain of 2 corresponds to a 50% reduction in page load time, the goal to which SPDY aspires.

As stated above, the impact that the website configuration has on the achieved gain will be examined first.

Table 5 shows the 3x3 matrix of website configurations. For each website, the maximum, minimum and mean gain values (across all channel conditions) are provided, in the manner shown below.

max
mean
min

In Table 5, gain values that represent SPDY/2 achieving the goal of 50% (or more) reduction in page load time are shaded green, and gain values that represent SPDY/2 degrading the page load time are shaded red.

gain ≥ 2
gain < 1

Additionally, row-wise, column-wise, and overall statistics are calculated and shown around the periphery of the 3x3 matrix.

Table 5. Website Impact on SPDY Gain

	1 server	2 servers	4 servers	average
Page A	4.1 **2.4** 1.6	1.8 **1.3** 1.1	1.8 **1.3** 1.1	4.1 **1.7** 1.1
Page B	3.3 **1.7** 0.8	1.4 **1.0** 0.6	1.4 **1.0** 0.7	3.3 **1.3** 0.6
Page C	1.0 **0.5** 0.3	1.0 **0.7** 0.4	1.1 **0.9** 0.5	1.1 **0.7** 0.3
average	4.1 **1.6** 0.3	1.8 **1.0** 0.4	1.8 **1.1** 0.5	4.1 **1.2** 0.3

These results show that SPDY worked well for Page A (many smaller images), showing gains across all test conditions and an average gain of 1.7. For Page B (many images more typical size), the results were a bit more hit-or-miss, with SPDY not always resulting in improved performance. Nonetheless, on average a gain of 1.3 was achieved. Page C (small number of large images), on the other hand, resulted in almost universally worse performance with SPDY than with HTTPS. In the worst case, SPDY resulted in a 3.3x increase in page load time (gain of 0.3). On average SPDY took 1.4x longer (gain of 0.7) to download Page C than traditional HTTPS.

In general, the SPDY impact (positive or negative) diminished as more servers were utilized. This is the result of the increased parallelism and decreased number of objects requested per SPDY session, which if taken to the extreme would result in a pattern of requests that is similar to the HTTPS case.

The results shown in Table 6 examine the impact that channel conditions have on SPDY performance (relative to HTTPS). Since the channel conditions were formed from a 2x2x2 matrix of data-rate, RTT, and packet-loss rate (PLR), the results are depicted as two 2x2 matrices of data-rate and RTT, one corresponding to the 0% PLR test cases, and the other corresponding to the 1% PLR test cases. For each channel condition, the results for all nine websites are summarized via the maximum, mean, and minimum gain achieved. Similar to the row and column statistics provided in the website impact analysis in Table 5, summary statistics for all three dimensions are provided around the periphery of the 2x2x2 cube (i.e., in the right-most column, the last row, and the third table).

Table 6. Channel Impact on SPDY Gain

0% PLR	20ms	100ms	average
	2.1	4.1	4.1
10/1	**1.4**	**1.7**	**1.6**
	0.7	0.8	0.7
	1.6	3.4	3.4
20/2	**1.2**	**1.6**	**1.4**
	0.5	0.7	0.5
	2.1	4.1	4.1
average	**1.3**	**1.6**	**1.5**
	0.5	0.7	0.5

1% PLR	20ms	100ms	average
	2.3	2.2	2.3
10/1	**1.1**	**0.9**	**1.0**
	0.4	0.3	0.3
	2.0	1.7	2.0
20/2	**1.0**	**0.8**	**0.9**
	0.3	0.3	0.3
	2.3	2.2	2.3
average	**1.0**	**0.8**	**0.9**
	0.3	0.3	0.3

	20ms	100ms	average
10/1	2.3 **1.3** 0.4	4.1 **1.3** 0.3	4.1 **1.3** 0.3
20/2	2.0 **1.1** 0.3	3.4 **1.2** 0.3	3.4 **1.1** 0.3
average	2.3 **1.2** 0.3	4.1 **1.2** 0.3	4.1 **1.2** 0.3

The most interesting observation here is the comparison between the two PLR tests. SPDY provided significant gains when the PLR was 0% (average gain 1.5), but showed worse average performance than HTTPS when packet loss was 1% (average gain 0.9). This effect was more pronounced in the large RTT cases as compared to the small RTT.

This result points to a fundamental weakness of SPDY. With a typical HTTPS download of a web page, the browser will open a large number of simultaneous TCP connections. In this case, a random packet loss will cause the one affected connection to temporarily reduce its congestion window (and hence the effective data rate), but the other connections will be unaffected, and in fact may be able to opportunistically make use of the bandwidth made available by the affected connection. The result for HTTPS is that random packet loss has only a minor impact on page download time. In the case of SPDY, the number of parallel TCP connections is dramatically reduced (by as much as a factor of six), so that random packet loss has a much bigger impact on the overall throughput.

In the absence of packet loss, SPDY provides better gains as the RTT increases. This is the result of reducing the number of round trips needed to fetch all of the page resources. Nonetheless, there were test cases with 0 packet loss where SPDY performed worse than HTTPS. This only occurred with Page C; SPDY always provided a benefit for Pages A & B when there was no packet loss. The root cause of the degradation is unknown.

Additionally, SPDY generally shows more gain in the lower Data Rate cases.

The overall average gain of 1.2 seen in our experiments aligns well with the performance gains that Google is seeing in their live deployments. In results presented in a December 8, 2011, Google TechTalk [http://www.cnx-software.com/2012/01/page/2/], they report a 15.4% improvement in page load time (equivalent to a gain of 1.18).

SPDY/2 provided an average gain of 1.2 (17% reduction in page load time)

5.4.2 Case 2: initcwnd=10 vs. initcwnd=3

The initcwnd increase provides very modest gain across the majority of test cases. In a few cases there was a marginal degradation of performance compared to the default initcwnd case. Across all test

conditions an average gain of 1.1 (or 9% reduction in page load time) was seen, as indicated in Table 7 and Table 8.

Table 7. Website Impact on Initcwnd Gain

	1 server	2 servers	4 servers	average
Page A	1.1 **1.0** 0.9	1.1 **1.0** 1.0	1.1 **1.0** 0.9	1.1 **1.0** 0.9
Page B	1.1 **1.0** 1.0	1.3 **1.1** 1.0	1.2 **1.0** 0.9	1.3 **1.0** 0.9
Page C	1.2 **1.1** 1.0	1.3 **1.1** 1.0	1.5 **1.2** 1.0	1.5 **1.1** 1.0
average	1.2 **1.1** 0.9	1.3 **1.1** 1.0	1.5 **1.1** 0.9	1.5 **1.1** 0.9

Overall the Page C test cases showed a slightly higher gain over the other two pages. Since Page C involves large files, it isn't surprising that the effect is slightly more pronounced than in Page A. In Page A, the majority of resources can be delivered in 3 packets or less anyway, so the increase in initcwnd doesn't reduce the number of round-trips.

Table 8. Channel Impact on Initcwnd Gain

0% PLR	20ms	100ms	average
10/1	1.1 **1.0** 1.0	1.2 **1.1** 1.0	1.2 **1.0** 1.0
20/2	1.1 **1.0** 1.0	1.1 **1.0** 0.9	1.1 **1.0** 0.9
average	1.1 **1.0** 1.0	1.2 **1.0** 0.9	1.2 **1.0** 0.9

1% PLR	20ms	100ms	average
10/1	1.5 **1.1** 0.9	1.2 **1.1** 1.0	1.5 **1.1** 0.9
20/2	1.4 **1.1** 0.9	1.3 **1.1** 0.9	1.4 **1.1** 0.9
average	1.5 **1.1** 0.9	1.3 **1.1** 0.9	1.5 **1.1** 0.9

	20ms	100ms	average
10/1	1.5 **1.1** 0.9	1.2 **1.1** 1.0	1.5 **1.1** 0.9
20/2	1.4 **1.1** 0.9	1.3 **1.1** 0.9	1.4 **1.1** 0.9
average	1.5 **1.1** 0.9	1.3 **1.1** 0.9	1.5 **1.1** 0.9

When viewing the results broken down by channel condition, we don't see a significant difference in gain from one channel condition to the next. Notably, even in the high RTT cases we don't see much change in the gain. This is likely due to the fact that the majority of resources had sizes that either fell below the 3 packet default initcwnd, or were much larger (i.e., over 210 packets each for the images in Page C) so that the initcwnd change might save a single RTT for a transfer that took 9 RTTs using the default initcwnd.

The initcwnd increase provided an average gain of 1.1 (9% reduction in page load time)

5.4.3 Case 3: SPDY+Initcwnd=10 vs. HTTPS+Initcwnd=3

The combination of SPDY and the initcwnd increase resulted in a benefit that was more than the product of the two individual gains. This is a result of the fact that, due to SPDY, all of the TCP sessions involved multiple round-trips, so the acceleration of initial data rate provided by the initcwnd increase resulted in tangible benefits. In fact, one case (Page A, single server) saw an astounding gain of 4.7. By examining the raw data in Appendix A.3, the data shows that this combination reduced the median page load time of 8.7 seconds down to 1.9 seconds. Unfortunately, this result is not typical, and Page C again experienced worse performance when the new protocol options are used. The overall average gain seen was 1.4 across all test cases as shown in Table 9 and Table 10.

Table 9. Website Impact on SPDY+Initcwnd Gain

	1 server	2 servers	4 servers	average
Page A	4.7 **3.2** 2.2	2.3 **1.5** 1.2	1.5 **1.4** 1.2	4.7 **2.0** 1.2
Page B	3.3 **1.9** 0.9	1.6 **1.1** 0.6	1.5 **1.1** 0.7	3.3 **1.4** 0.6
Page C	1.0 **0.6** 0.3	1.0 **0.7** 0.4	1.1 **0.9** 0.6	1.1 **0.7** 0.3
average	4.7 **1.9** 0.3	2.3 **1.1** 0.4	1.5 **1.1** 0.6	4.7 **1.4** 0.3

Table 10. Channel Impact on SPDY+Initcwnd Gain

0% PLR	20ms	100ms	average
10/1	3.2 **1.7** 1.0	4.7 **1.8** 0.9	4.7 **1.8** 0.9
20/2	3.1 **1.6** 0.7	3.9 **1.7** 0.7	3.9 **1.6** 0.7
average	3.2 **1.7** 0.7	4.7 **1.7** 0.7	4.7 **1.7** 0.7

1% PLR	20ms	100ms	average
10/1	3.1 **1.2** 0.4	2.4 **0.9** 0.3	3.1 **1.1** 0.3
20/2	2.8 **1.2** 0.3	2.2 **1.0** 0.3	2.8 **1.1** 0.3
average	3.1 **1.2** 0.3	2.4 **1.0** 0.3	3.1 **1.1** 0.3

	20ms	100ms	average
10/1	3.2 **1.5** 0.4	4.7 **1.4** 0.3	4.7 **1.4** 0.3
20/2	3.1 **1.4** 0.3	3.9 **1.3** 0.3	3.9 **1.3** 0.3
average	3.2 **1.4** 0.3	4.7 **1.3** 0.3	4.7 **1.4** 0.3

In the packet loss test cases, there was a fairly significant performance loss with SPDY, as reported in earlier test cases. This is particularly true in the high RTT cases.

SPDY/2 with the initcwnd increase provided an average gain of 1.4 (29% reduction in page load time)

5.4.4 Results Summary

The results are summarized further in Table 11. A caveat on the average results presented here is worth noting: the 72 test cases were selected to test the protocol changes in a fairly wide range of conditions in order to understand the impact that individual factors have on the performance gains. As a result, the 72 test cases likely do not comprise a statistically accurate random sampling of real-world sites, and so the average results presented here may not accurately reflect the average performance gain that would be seen in the real world. However, as noted previously, the average results do appear to align well with the average gains seen by Google via their live deployments with users utilizing the Chrome browser. Nonetheless, it is also interesting to examine the average gain achieved in a particular subset of the test conditions. For that we select the subset consisting of pages A&B operating on a wireline network (PLR=0%). For that subset, we see that the combination of SPDY/2 and the initcwnd increase achieves an average gain of 2.1 (52% reduction in page load time).

Table 11. Summary of Results

	Case 1 SPDY	Case 2 initcwnd	Case 3 SPDY+initcwnd
Best Gain (all test conditions)	4.1 (A,1,10,100,0)	1.5 (C,4,10,20,1)	4.7 (A,1,10,100,0)
Average Gain (all test conditions)	1.2	1.1	1.4
Worst Gain (all test conditions)	0.3 (C,1,x,x,1)	0.9 (A,1,20,100,0)	0.3 (C,1,20,100,1)
# test cases achieving gain ≥ 1	40 of 72 (56%)	58 of 72 (81%)	44 of 72 (61%)
# test cases achieving gain ≥ 2	8 of 72 (11%)	0 of 72 (0%)	14 of 72 (19%)
Average Gain (Pages A&B with PLR=0%)	1.8	1.0	2.1

In the absence of packet loss, the combination of SPDY/2 and the increased initcwnd can achieve the goal of 50% reduction in page load time for certain web pages.

6 RECOMMENDATIONS AND CONCLUSIONS

The results presented here indicate that SPDY/2 in conjunction with the TCP Initial Congestion Window increase has the potential to improve or impair web page download performance depending on a number of factors.

Deployment of SPDY/2 for general-purpose web servers should be considered in light the following concerns:

1. While the Chrome browser is an important browser (approximately 25% market share as of Jan. 2012), it is the only browser that truly utilizes SPDY out of the box at the time of writing this report.
2. Some web page downloads were significantly impaired by SPDY.
3. Work is underway on a revision to the protocol, and with standardization in the IETF a possibility in the near future, waiting may be worthwhile to see what develops in this space.

On the other hand, for a controlled environment where the MSO provides the servers, the client software, and the web content, SPDY might be a valuable tool. An example application might be a remote user interface (RUI) that is delivered via HTTP to an MSO-controlled CPE device, e.g., to support a video service. Some specific use-cases include: an MSO-provided set-top box that utilizes an HTML-based Remote UI; and an MSO-provided Gateway device that caches user interface elements to serve to Internet Protocol Set-Top Boxes (IP-STBs), tablets, etc., in the home. Note that the basic features of SPDY are not expected to provide any tangible benefits as compared to HTTP for video streaming (e.g., HTTP Live Streaming, MS Smooth Streaming, MPEG-DASH). In those cases the client makes a series of requests for the video fragment files that make up the video program. Since multiple fragments are not requested simultaneously, the multiplexing feature of SPDY would not be utilized. Nor would there be any gains from request header compression. The SPDY server-push function could potentially be used to push fragment files into a local (e.g., Gateway) cache in advance of them being requested by the client, and this technique could be utilized to enable centralized control over video stream bitrate adaptation. But, a similar outcome could be accomplished more simply via an HTTP/1.1 PUT or POST method.

In addition to potential uses for SPDY by cable MSOs, another area of interest might be the impact that SPDY would have on network traffic if it were to be widely adopted. In general the story here is good, by reducing the number of simultaneous TCP sessions and extending the duration of many of the sessions, other applications could see improved performance as a result of TCP congestion avoidance being invoked far more often. Secondly, the expectation would be to see a slight reduction in data usage, due to the greater efficiency that SPDY provides (fewer TCP control packets), as well as a slight increase in average packet size, due to the multiplexing of HTTP responses. Both of these factors will serve to reduce the packets per second rate of the network, which improves scalability of CMTSs, routers, DPI boxes, etc. Finally, equipment implementing Carrier Grade NAT (which will be a key element of the IPv6 transition in some networks) could see improved scalability as the number of simultaneous TCP connections is reduced.

In regards to the increase in the TCP Initial Congestion Window, while we see only marginal gains resulting from this enhancement, the change can be made simply by changing a server operating system parameter. It requires no client modifications. As a result, we see no reason not to set the server initcwnd at the proposed value of 10.

APPENDIX A RAW DATA

A.1 Ubuntu Raw Data

Website Configuration		Channel Condition			Median Page Load Time (ms)	
Page	Num Servers	DS (Mbps)	RTT (ms)	PLR	HTTPS, initcwnd=def	SPDY, initcwnd=def
A	1	10	100	1%	10771	5074
A	2	10	100	1%	7697	6595
A	4	10	100	1%	5929	4787
B	1	10	100	1%	12520	13453
B	2	10	100	1%	7968	10613
B	4	10	100	1%	5559	7198
C	1	10	100	1%	8468	28363
C	2	10	100	1%	5711	15316
C	4	10	100	1%	5588	10410

A.2 WinXP Raw Data

Website Configuration		Channel Condition			Median Page Load Time (ms)	
Page	Num Servers	DS (Mbps)	RTT (ms)	PLR	HTTPS, initcwnd=def	SPDY, initcwnd=def
A	1	20	100	0%	8645	3625
A	2	20	100	0%	5822	4650
A	4	20	100	0%	4309	3611
B	1	20	100	0%	10557	4707
B	2	20	100	0%	6511	5386
B	4	20	100	0%	4339	4361
C	1	20	100	0%	4537	7329
C	2	20	100	0%	3084	4872
C	4	20	100	0%	3342	3892
A	1	20	100	1%	11879	6004
A	2	20	100	1%	8620	7145
A	4	20	100	1%	6642	5448
B	1	20	100	1%	14815	17546
B	2	20	100	1%	9581	12625
B	4	20	100	1%	6373	7975
C	1	20	100	1%	10641	35263
C	2	20	100	1%	7748	19906
C	4	20	100	1%	7656	13092
A	1	10	100	0%	8644	4012
A	2	10	100	0%	5786	4675
A	4	10	100	0%	4221	3698

Website Configuration		Channel Condition			Median Page Load Time (ms)	
Page	Num Servers	DS (Mbps)	RTT (ms)	PLR	HTTPS, initcwnd=def	SPDY, initcwnd=def
B	1	10	100	0%	10557	4325
B	2	10	100	0%	6414	5566
B	4	10	100	0%	4358	4548
C	1	10	100	0%	4830	7488
C	2	10	100	0%	3138	5084
C	4	10	100	0%	3249	4172
A	1	10	100	1%	11537	5173
A	2	10	100	1%	8179	6886
A	4	10	100	1%	6201	5116
B	1	10	100	1%	14772	15624
B	2	10	100	1%	9388	12406
B	4	10	100	1%	6232	7467
C	1	10	100	1%	9573	30003
C	2	10	100	1%	7581	18297
C	4	10	100	1%	7606	12496

A.3 Win7 Raw Data

Website Configuration		Channel Condition			Median Page Load Time (ms)			
Page	Num Servers	DS (Mbps)	RTT (ms)	PLR	HTTPS, initcwnd=def	SPDY, initcwnd=def	HTTPS, initcwnd=10	SPDY, initcwnd=10
A	1	20	20	0%	2651	1618	2642	854
A	2	20	20	0%	1881	1175	1774	887
A	4	20	20	0%	1755	1262	1695	1207
B	1	20	20	0%	4441	4084	4435	2063
B	2	20	20	0%	2991	2398	2951	2041
B	4	20	20	0%	2466	1979	2376	1694
C	1	20	20	0%	2373	5112	2180	3261
C	2	20	20	0%	2175	2644	2269	2195
C	4	20	20	0%	2215	2278	2198	2125
A	1	20	100	0%	7341	2184	8387	1894
A	2	20	100	0%	5517	4471	5471	3990
A	4	20	100	0%	4338	3492	4171	3169
B	1	20	100	0%	9382	2939	9867	3082
B	2	20	100	0%	6503	5112	6275	5541
B	4	20	100	0%	5292	4125	5440	3880
C	1	20	100	0%	4740	4893	4461	5085
C	2	20	100	0%	4161	5669	3677	5591
C	4	20	100	0%	4313	4697	3780	4373

Website Configuration		Channel Condition			Median Page Load Time (ms)			
Page	Num Servers	DS (Mbps)	RTT (ms)	PLR	HTTPS, initcwnd=def	SPDY, initcwnd=def	HTTPS, initcwnd=10	SPDY, initcwnd=10
A	1	20	20	1%	4213	2091	4271	1499
A	2	20	20	1%	18428	14762	18600	13693
A	4	20	20	1%	14862	12015	15398	11542
B	1	20	20	1%	6811	5776	6682	5049
B	2	20	20	1%	29509	33087	23488	29776
B	4	20	20	1%	18085	25951	19745	18999
C	1	20	20	1%	3509	11767	2985	11037
C	2	20	20	1%	22234	47691	20874	47068
C	4	20	20	1%	28099	32248	20600	26658
A	1	20	100	1%	11024	6300	10607	5092
A	2	20	100	1%	30155	26138	28402	24695
A	4	20	100	1%	24625	22404	21919	16357
B	1	20	100	1%	14352	17765	13116	15681
B	2	20	100	1%	36520	60658	32067	59662
B	4	20	100	1%	24893	35236	27591	31443
C	1	20	100	1%	10311	33026	8927	33167
C	2	20	100	1%	41841	101007	31538	100423
C	4	20	100	1%	37827	61020	31629	58993
A	1	10	20	0%	2854	1638	2611	885
A	2	10	20	0%	2112	1185	1968	932
A	4	10	20	0%	2355	1289	2315	1533
B	1	10	20	0%	4875	2324	4508	2077
B	2	10	20	0%	3469	2547	3346	2165
B	4	10	20	0%	3117	2312	3215	2128
C	1	10	20	0%	3356	4994	3247	3511
C	2	10	20	0%	3258	3272	3215	3141
C	4	10	20	0%	3249	3232	3247	3146
A	1	10	100	0%	8692	2126	8417	1853
A	2	10	100	0%	5573	4466	5696	4060
A	4	10	100	0%	4633	3598	4405	3405
B	1	10	100	0%	10301	3126	10278	3091
B	2	10	100	0%	6762	5890	6730	5503
B	4	10	100	0%	5615	3959	5386	4114
C	1	10	100	0%	4850	6012	4533	5104
C	2	10	100	0%	5480	6162	4741	5965
C	4	10	100	0%	5467	4987	4798	4817
A	1	10	20	1%	4523	2000	4057	1457
A	2	10	20	1%	17564	12570	15869	12810

Website Configuration		Channel Condition			Median Page Load Time (ms)			
Page	Num Servers	DS (Mbps)	RTT (ms)	PLR	HTTPS, initcwnd=def	SPDY, initcwnd=def	HTTPS, initcwnd=10	SPDY, initcwnd=10
A	4	10	20	1%	13525	12200	14491	9767
B	1	10	20	1%	6998	4819	6140	4751
B	2	10	20	1%	27637	32944	28855	31193
B	4	10	20	1%	20083	22505	17064	19742
C	1	10	20	1%	3822	10477	3582	9801
C	2	10	20	1%	24029	49178	23001	47226
C	4	10	20	1%	25864	29327	17777	30610
A	1	10	100	1%	11049	5124	10655	4636
A	2	10	100	1%	29747	27242	27351	22603
A	4	10	100	1%	23708	19451	20964	19999
B	1	10	100	1%	14416	16466	13113	15309
B	2	10	100	1%	34631	62502	34802	55112
B	4	10	100	1%	23805	35175	22913	31825
C	1	10	100	1%	10042	30557	8767	31672
C	2	10	100	1%	38520	92986	31824	90527
C	4	10	100	1%	32947	63725	30410	56366
C	4	10	100	1%	7454	12343	5607	11048