A comparison of SPDY and HTTP performance

Jitendra Padhye and Henrik Frystyk Nielsen.
July 26, 2012.

Table of Contents

Introduction

We compare the performance of SPDY and HTTP in a variety of setting using a controlled tested study. We rely in publically available software and run with mostly default configurations. It is not our aim to extract every last bit of performance out of either SPDY or HTTP. Rather, we are interested in understanding how various mechanisms used by the two protocols affect page load times. This report documents our preliminary findings, and is being released to engender discussion. We continue to perform more detailed tests.

Testbed

Our testbed consists of two machines, connected to each other via a 1Gbps Ethernet link. One machine functions as the client (i.e. runs the browser), while other functions as a server.

Hardware

Both machines have modern multicore CPUs and several gigabytes of RAM. The hardware was not a bottleneck in any of the tests.

Software

Client:

- 64 bit Windows 7
- Chromimum browser, version 20.0.1132.57. [1]
- Chromedriver, version 20.0.1133. [2]
- Network Emulator for Windows (similar to dummynet), version 10.0.0003.1. [3]
- Acrylic DNS Proxy, version 0.9.19. [4]
- NetMon 3.4 to capture packet traces. [5]

Server:

- 32 bit Fedora Linux 3.4.2-1.
- Apache server, 2.2.22.
- mod_spdy, version 2.
- mod_pagespeed
- Dummynet (for some experiments)

Methodology

For some of the experiments, we use a dummy web page consisting of several images and CSS files. The composition of the dummy web page is shown in Table 1. This page is not meant to be a representative page in any way – in fact, it is constructed specifically to highlight certain aspects of performance, as we shall see later.

File	Size in Bytes
Index.html	14KB
20 images	1.1MB
3 style sheets	1.3KB

Table 1: Composition of dummy web page

For other experiments, we copied and archived the contents of popular web sites using WinHTTrack [6]. WinHTTrack converts local links on these web pages to relative links, while leaving non-local links intact. The conversion is not foolproof. For example, WinHTTrack will not capture URLs constructed on-the-fly by javascripts on the page. To prevent such requests from leaving the testbed, we use the Acrylic DNS proxy [4] to ensure that all host names resolve to the IP address of the testbed server. The server will return a 404 error for any objects that it does not have.

The browser is driven by a simple C# program that controls browser via Chromedriver using RemoteWebDriver. All experiments are performed with a cold browser cache. Network emulator and Dummynet are used to emulate different bandwidth and link delays.

Our performance metric is the page load time. It is well known that this metric is hard to define and measure correctly. In our tests, we rely on the times reported by the Chromedriver. The browser is instructed to fetch Navigate() command. The page is assumed to be loaded when the Navigate() call returns.

In the results reported below, each data point is a median of five trials, unless otherwise noted.

Results

Dummy web page

For experiments in this section, we use the dummy web page shown in **Error! Reference source not found.**. We simulate a variety of link bandwidths and delay as described below.

We begin by simulating a link with 10mbps bandwidth, and a variety of round trip delays. We use HTTP and SPDY with and without SSL. Figure 1 shows the results. For HTTP we used default Apache settings, which disables keepalive, and no compression is used. We see that SPDY performs significantly better than HTTP, especially at higher RTTs. For example, at 100ms, SPDY reduces page load time by 39%. Header compression has been mentioned as one of the important features of SPDY. However, in this example, header compression does not have much of an impact. With HTTP, each page download consumes 1.222MB total, including all packet-level headers. With SPDY, the number of bytes consumed is 1.218MB. The primary reason behind HTTP's poor performance is lack of keepalive, which forces the browser to open a new connection for each object (a total of 24).

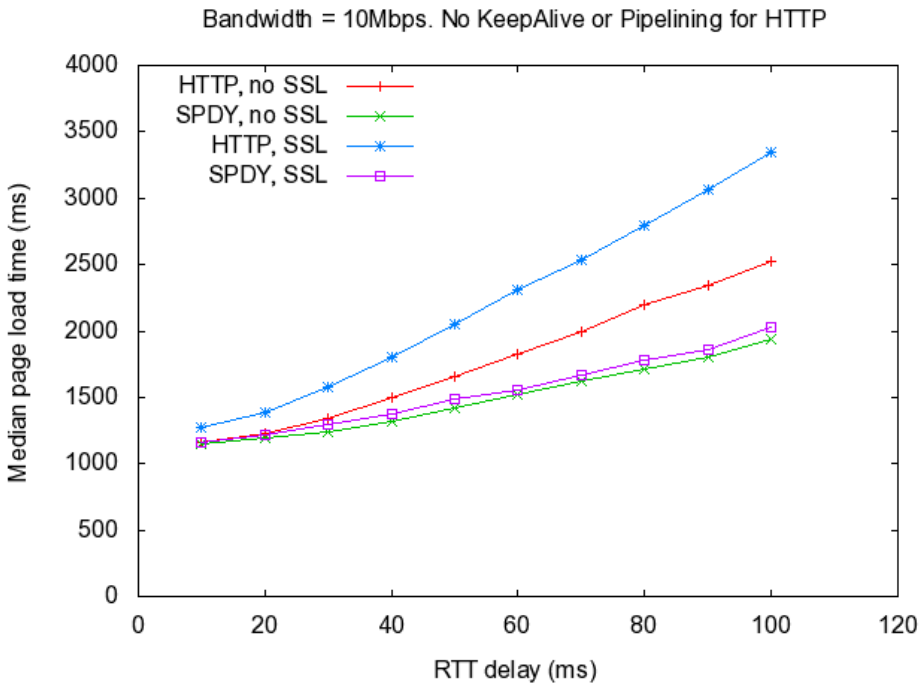

Figure 1: Dummy web page, 10Mbps

Next, we repeat the experiment with a 1Mbps link. The results are shown in Figure 2. In this setting, the low bandwidth of the link the dominant factor affecting the page load time. As a result, SPDY provides only 3-8% improvement over HTTP in this setting.

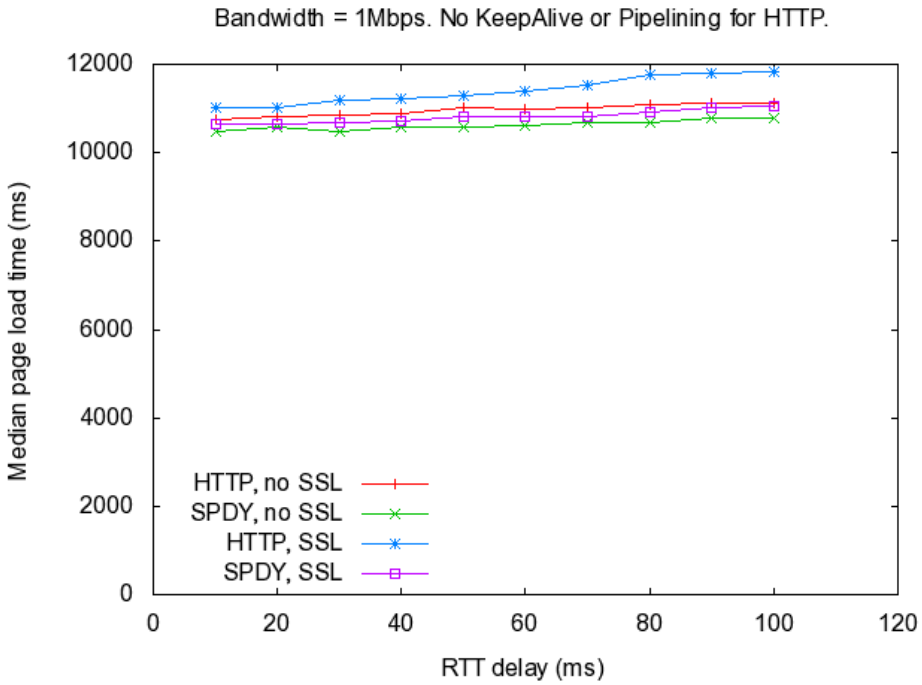

Figure 2: Dummy web page, 1Mbps

In Figure 1, we saw that HTTP performs quite poorly, as it opened a new connection for fetching each object. We now set turn KeepAlive on in Apache configuration, and allow Chrome to use pipelining. With this change, the HTTP's performance improves dramatically, as shown in Figure 3.

The performance of HTTP improves even further, if we turn on mod_pagespeed, which minifies and inlines some of the images and javascripts. With these changes, HTTP opens only 4 to 6 connections for page download, instead of 24. The number of bytes transferred per page load is also reduced to 997KB, on average. The performance of HTTP with pipelining and minification is very close (and slightly better) than SPDY – even with minification.

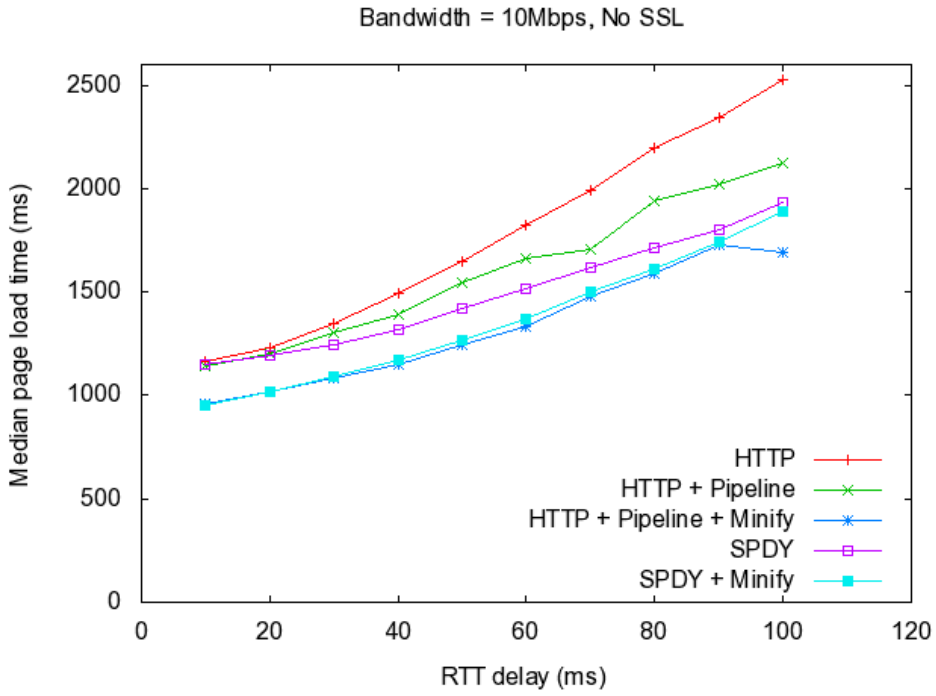

Figure 3: Impact of pipelining and minification

Archived Websites

The dummy web page was an artificial construct. To see how SPDY and HTTP compare when used with real websites, we use contents from real websites, copied and archived using WinHTTrack, as describe earlier.

We begin with the mobile version of the CNN home page, located at http://m.cnn.com. While fetching the web page, the user agent string was set to simulate a Windows Phone (Nokia Lumia 900).

The CNN mobile web page is optimized for mobile experience. It consist of 11 objects (1 html, 1 CSS and 9 images), totaling only 38.5KB. The style sheet is fetched from a different domain (i.cdn.turner.com), but we served it from the testbed server along with the rest of the webpage. This allows SPDY to aggregate the fetching of this style sheet on the same TCP connection. The there are no javascripts on the webpage generating dynamic fetches. As a result, our local server is able to satisfy all requests and no 404s are generated.

As before, we start with a 10Mbps link between the client and the server. It may seem strange to fetch a mobile-specific website using a 10Mbp slink, but we note that the upcoming LTE standard promises download speeds of up to 300Mbps. The results are shown in Figure 4. HTTP is used with Keepalive and pipelining. Given the small size of the web page, and the ample link bandwidth, it is no surprise that performance of SPDY and HTTP are quite similar. We note that SPDY appears to perform slightly worse than HTTP at lower round trip times. We are investigating this issue further.

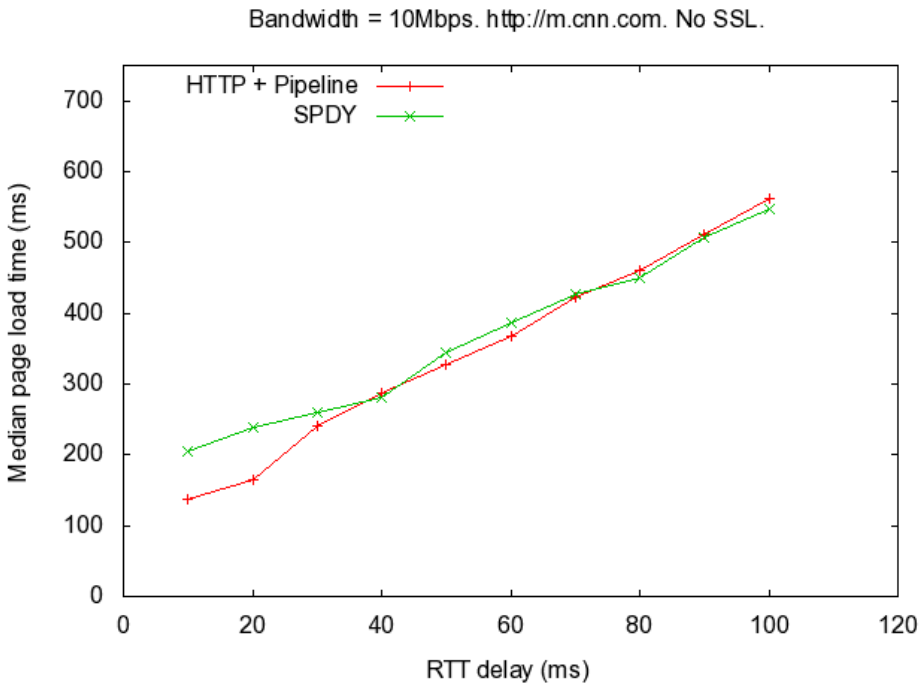

Figure 4: CNN Mobile, 10Mbps.

We then repeat the test with 1Mbps link. The results are shown in Figure 5. We see that without minification, SPDY does better than HTTP. The reduction in page load time is between 9-14%. The improvement comes from two factors. First, HTTP opens multiple connections to download the 11 objects on the web page, while SPDY opens only one. Second, SPDY transfers less data per page download. HTTP transfers, on average, 42KB during the download, while SPDY transfers only 39.9KB, due to header compression and other optimizations.

However, once we enable page minification via mod_pagespeed, performance of HTTP improves dramatically. Due to image inlining and compression, the number of bytes transferred per page download is reduced to just 13.5KB on average, and each transfer uses just a single TCP connection. Only two HTTP request/responses are required; one to download the index page with inlined images, and other to download the CSS style sheet.

We are currently unable to explain the spike in page load time at 70ms delay. Packet logs indicate that the server is consistently late in replying to the request for the style sheet at this delay. We are investigating this further.

We also see that the performance of SPDY with mod_pagespeed takes 20-47% more to load the page than HTTP with mod_pagespeed, except at 70ms. This is because SPDY transfers more bytes (24.5KB) per page download than HTTP. Evidently, mod_pagespeed does not apply all possible optimizations when used with SPDY. We are investigating this further.

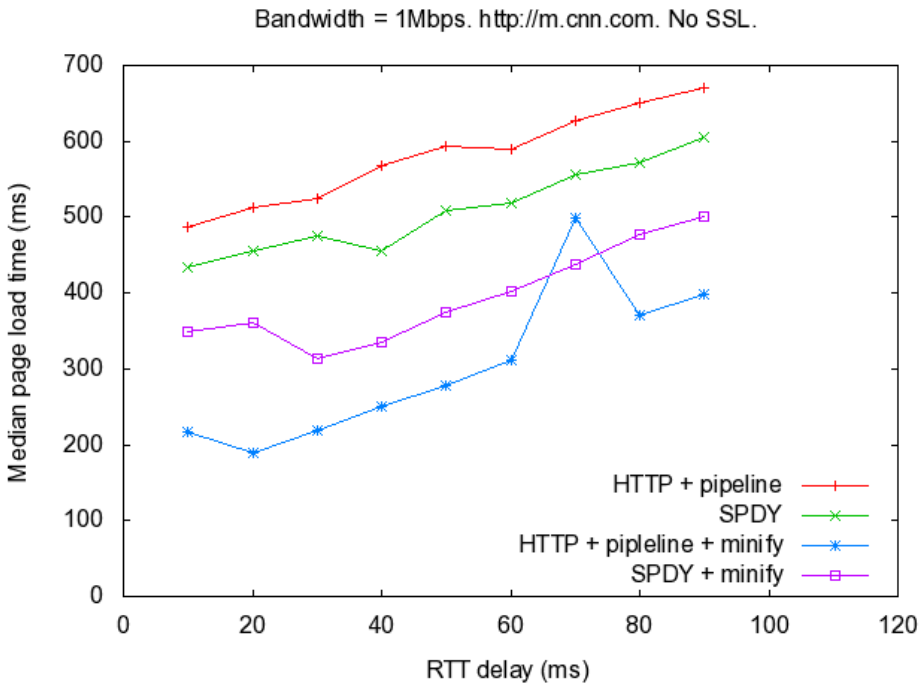

Figure 5: CNN Mobile, 1Mbps

Recall that our tests with the dummy web page showed that SSL has minimal impact on SPDY's performance (see Figure 1 and Figure 2). However, the CNN web page is much smaller than the dummy web page, and we would expect SSL to have more impact on the page load time. To test this hypothesis, we repeated the test in Figure 5 with SSL. The results are shown in Figure 6. We see that in this scenario, SSL can increase the page load time by up to 20%. SSL impacts performance of HTTP in a similar manner.

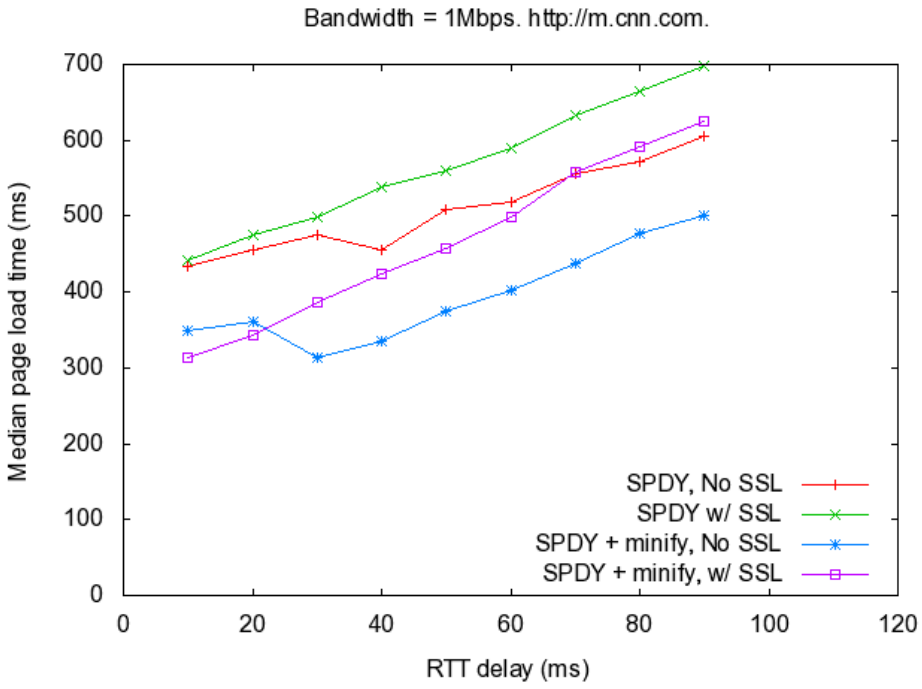

Figure 6: Impact of SSL on SPDY performance

We repeated the tests above with additional websites, shown in Table 2. The websites labeled "mobile" were copied via HTTtrack using a user agent string that was set to simulate a Windows Phone (Nokia Lumia 900). We tested these websites by emulating a

1Mbps link, with 150ms round trip delay. The non-mobile websites were fetched with default IE user agent string. These websites were tested with a 10Mbps link, and 150ms round trip delay. The results are shown in Figure 7 and Figure 8.

Mobile Websites	Non-Mobile
m.cnn.com	bbc.com
m.cnet.com	greenweddingshoes.com
m.espncricinfo.com	developers.goole.com/speed/articles/spdy-for-mobile.html
m.politico.com	

Table 2: Selected Websites

In Figure 7, we see that without SSL, HTTP with pipelining and minification performs better than SPDY. With SSL, HTTP performance suffers significantly, due to the high round trip delay. SSL also increases the page load time with SPDY by 15-25%. In Figure 8, we see similar results. We have only shown the performance of SPDY with SSL in this graph, because the browser fails to render bbc and the greenweddingshoes websites, when SPDY is used without SSL. We are investigating this issue further.

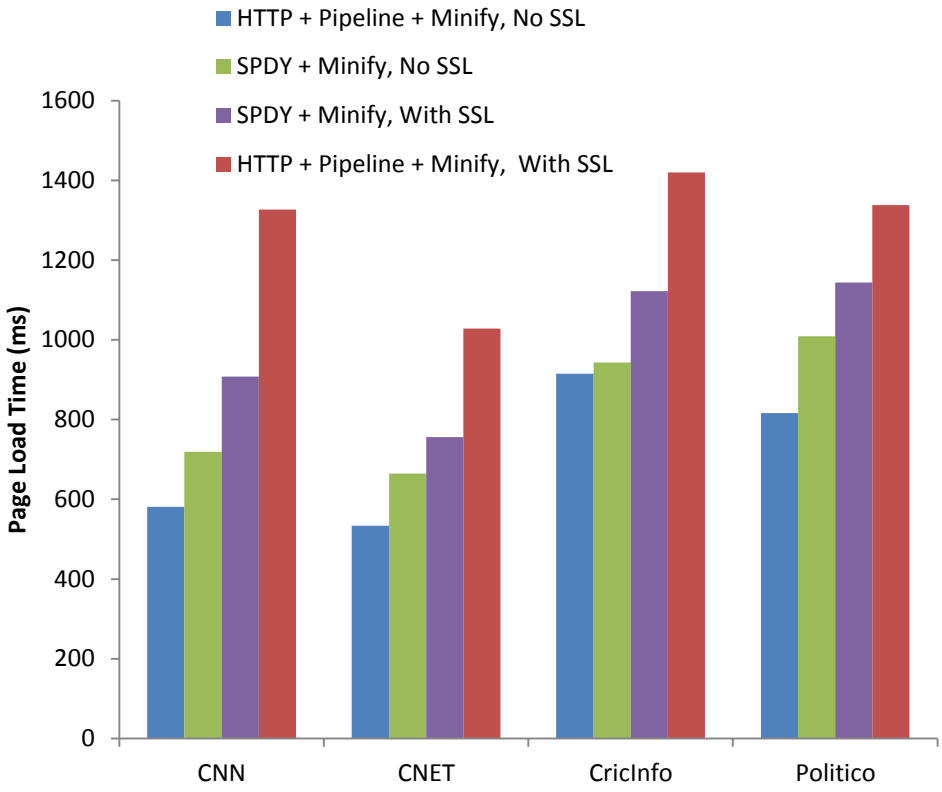

Figure 7: Mobile Websites. 1Mbps, 150ms.

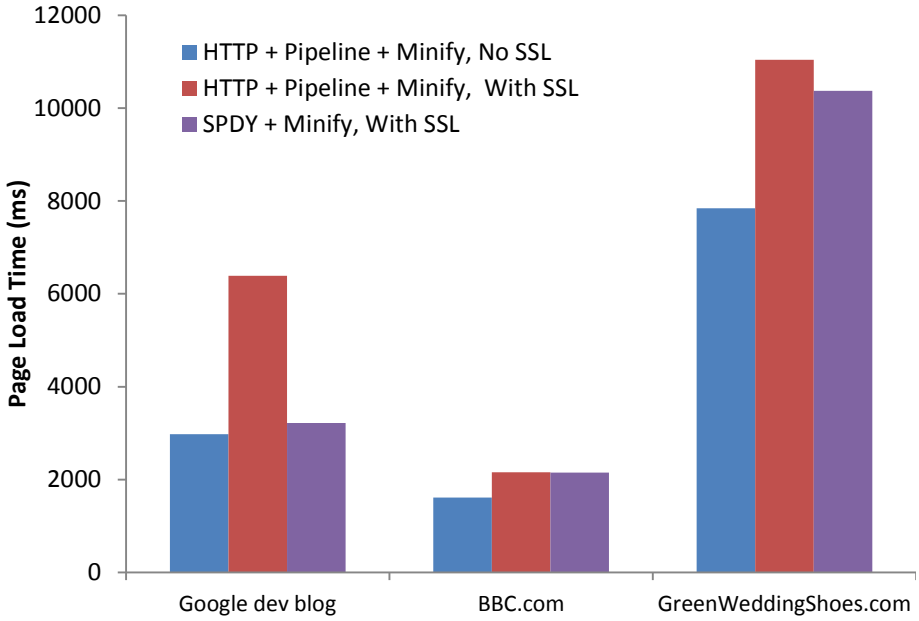

Figure 8: Non-Mobile websites. 10Mbps, 150ms.

Conclusion

We stress that these experiments are preliminary, and are primarily meant to (a) spur a discussion and (b) to demonstrate the need for more detailed analysis and testing. Our primary finding from the study is that judicious use of existing and well-understood optimizations such as pipelining and minification can bring HTTP's performance close to that of SPDY. We also find that for small web pages, the overhead of SSL handshakes can have a significant impact on SPDY's performance. Neither of these two findings are surprising and have been documented and/or articulated by others as well.

We plan to continue our testing. Among other things, we plan to study the impact of server push mechanism proposed in the SPDY standard, and tease out the impact of flow control and multiplexing mechanisms that are integral to the SPDY protocol. We also plan to investigate the impact of SSL on HTTP's performance in more detail.

Acknowledgements

A number of people helped in the preparation of this report. We would like to thank Henrik Frystyk Nielsen, Anton Pavlov, Daehyeok Kim, Sharad Agarwal, Adalberto Foresti, Rob Trace, Gabriel Montenegro and Xiao Sophia Wang for their help.

Bibliography

[1] [Online]. Available: https://www.google.com/intl/en/chrome/browser/.

[2] [Online]. Available: http://code.google.com/p/selenium/wiki/ChromeDriver.

[3] Microsoft, *Network Emulator for Windows - Internal only.*.

[4] [Online]. Available: http://sourceforge.net/projects/acrylic/.

[5] [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=4865.

[6] [Online]. Available: http://www.httrack.com/.