Evaluating Web-latency reducing Protocols in Mobile Environments

Usama Shamsher
Wang Xiao Jun

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden
This thesis is submitted to the School of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering. The thesis is equivalent to 20 weeks of full time studies.

This Master Thesis is typeset using \LaTeX

Contact Information:

Author(1):
Usama Shamsher
E-mail: usamashamsher@msn.com

Author(2):
Wang Xiao Jun
E-mail: vvoody.wang@gmail.com

University advisor:
Dr. David Erman
School of Computing

University Examiner:
Dr. Patrik Arlos
School of Computing

Contact Information:

School of Computing,
Department of Telecommunication Systems
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

Internet : www.bth.se/com
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
Abstract

User perceived latency is the most prominent performance issue influencing the World Wide Web (www) presently. Hyper-Text Transfer Protocol (HTTP) and Transmission Control Protocol (TCP) have been the backbone of web transport for decades, thus received a lot of attention recently due to end-to-end performance degradation in mobile environments. Inefficiencies of HTTP and TCP strongly affect web response time mainly in resource-limited devices. HTTP compression reduces some of the burden imposed by TCP slow start phase. However, compression is still an underutilized feature of the web today [1]. In order to fulfill the end user expectations, we can optimize HTTP to improve Page Load Time (PLT), low memory usage and better network utilization. SPDY, a web latency reducing protocol and HTTP pipelining are a recent proposal to provide faster information exchange over web.

Through the course of this work, we present a comprehensive study of new approaches to reduce mobile web latency. At first, we measure the PLT after implementing SPDY, HTTP and HTTP pipelining. Secondly, we also analyze the performance of these protocols after tuning the network parameters like bandwidth and round-trip time (RTT). Finally, we compare the performance of HTTP and other latency reducing protocols. We have conducted all experiments over DummyNet under user-configured network conditions. We critically discuss the challenges of shifting from HTTP to these latency-reducing protocols.

**Keywords:** DummyNet, Web-latency, Mobile Web Browsing, Page Load Time, HTTP, SPDY.
Acknowledgments

First of all, we would like to thank our supervisor Dr. David Erman for his guidance.

We would like to extend our special gratitude to Dr. Patrik Arlos for his valuable suggestions.

In the end, respect to our beloved parents, teachers and friends for their support through our life.

Usama Shamsher
Wang Xiao Jun

June, 2013
Dear reader, this document is the result of two year master degree in Electrical engineering with emphasize on Telecommunication Systems offered by Department of Telecommunication Systems, school of Computing at Blekinge Institute of Technology, Karlskrona, Sweden.

This thesis covers eight chapters which are briefed as follows:

**Chapter-1**
Chapter 1 discusses the importance of the work, motivation for the research topic and current challenges in mobile web browsing.

**Chapter-2**
Chapter 2 describes the previously conducted researches in this field till now.

**Chapter-3**
Chapter 3 explains the relevant technical background. This section provides knowledge about the different layers of mobile web browsing being practiced currently.

**Chapter-4**
Chapter 4 investigates the current protocols, positive and negatives from the standpoint of mobile web browsing. This section also proposes some improvements to overcome the deficiencies in current protocols.

**Chapter-5**
Chapter 5 describes experimental methodology that comprises network setup, traffic generation configurations, scenarios, metrics and the specification of Dummy Webpage used for this experiment.

**Chapter-6**
Chapter 6 compares the performance results of HTTP with SPDY and HTTP-Pipelining.

**Chapter-7**
Chapter 7 discusses the experience gained through this experiment.
Chapter-8
Chapter 8 sums up the conclusion and future work.

Appendix
Appendix covers information about every relevant technical work that has been done.
# Contents

Abstract i  
Acknowledgments ii  
Preface iii  
List of Figures vii  
List of Tables viii  
List of Abbreviations ix  

## 1 Introduction 1  
1.1 Motivation ...................................................................... 1  
1.2 Research Questions ..................................................... 3  
1.3 Research Methodology .................................................. 3  

## 2 Related Works 4  

## 3 Technical Background 6  
3.1 Web Browsing in Cellular Networks ......................... 6  
3.2 HTTP Requests and Responses ................................. 9  

## 4 A Survey of Current Protocols 10  
4.1 Deficiencies in Current Protocols ............................... 10  
4.2 Proposals for Faster Web ........................................... 11  
4.2.1 Header Compression ............................................. 11  
4.2.2 Request Prioritization .......................................... 12  
4.2.3 Multiplexing .......................................................... 12  

## 5 Methodology 15  
5.1 Testbed and Setup ....................................................... 15  
5.2 Network Setup ........................................................... 16  
5.3 Automation Testing Framework ................................. 16  
5.4 Client Side Configuration .......................................... 16  
5.5 Server Side Configuration .......................................... 17  
5.6 Hardware Configuration of Equipment ....................... 17  
5.7 Software Configuration of Equipment ....................... 17
List of Figures

3.1 Message flow in Mobile Web Browsing .................................. 8
4.1 HTTP Pipelining ............................................................. 13
4.2 Multiplexing ................................................................. 14
5.1 Basic Experimental System ............................................... 15
5.2 Basic Schematics ............................................................ 16
5.3 Automation Testing Framework .......................................... 19
6.1 Impact of SSL Handshake Delays on SPDY Performance ....... 22
6.2 Impact of SSL Handshake Delays on HTTP Performance ...... 23
6.3 SPDY Mode SSL, Download Bandwidth fixed at 10 Mbps ....... 24
6.4 HTTP Mode SSL, Download Bandwidth fixed at 10 Mbps ....... 24
6.5 Impact of SSL Handshakes on PLT ................................. 25
6.6 Impact of SSL on SPDY and HTTP Performance ............... 26
## List of Tables

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5</td>
<td>Domain Names and Distributed Contents</td>
<td>17</td>
</tr>
<tr>
<td>5.6</td>
<td>Hardware Configuration</td>
<td>17</td>
</tr>
<tr>
<td>5.7</td>
<td>Server side Software Configuration</td>
<td>18</td>
</tr>
<tr>
<td>5.7.1</td>
<td>Client side Software Configuration</td>
<td>18</td>
</tr>
<tr>
<td>5.8</td>
<td>Dummy Webpage Composition</td>
<td>19</td>
</tr>
<tr>
<td>C.1</td>
<td>SPDY mode SSL with Bandwidth fixed at 10 Mbps</td>
<td>41</td>
</tr>
<tr>
<td>C.2</td>
<td>SPDY mode Non-SSL with Bandwidth fixed at 10 Mbps</td>
<td>42</td>
</tr>
<tr>
<td>C.3</td>
<td>HTTP mode SSL with Bandwidth fixed at 10 Mbps</td>
<td>43</td>
</tr>
<tr>
<td>C.4</td>
<td>HTTP mode Non-SSL with Bandwidth fixed at 10 Mbps</td>
<td>44</td>
</tr>
<tr>
<td>C.5</td>
<td>SPDY mode SSL with RTT fixed at 150 ms</td>
<td>45</td>
</tr>
<tr>
<td>C.6</td>
<td>SPDY mode Non-SSL with RTT fixed at 150 ms</td>
<td>46</td>
</tr>
<tr>
<td>C.7</td>
<td>HTTP mode SSL with RTT fixed at 150 ms</td>
<td>47</td>
</tr>
<tr>
<td>C.8</td>
<td>HTTP mode Non-SSL with RTT fixed at 150 ms</td>
<td>48</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------</td>
<td></td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
<td></td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
<td></td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
<td></td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
<td></td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
<td></td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
<td></td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
<td></td>
</tr>
<tr>
<td>HTTPS</td>
<td>Hypertext Transfer Protocol Secure</td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
<td></td>
</tr>
<tr>
<td>ISPs</td>
<td>Internet Service Providers</td>
<td></td>
</tr>
<tr>
<td>PLT</td>
<td>Page Load Time</td>
<td></td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
<td></td>
</tr>
<tr>
<td>DOS</td>
<td>Denial of Service</td>
<td></td>
</tr>
<tr>
<td>DDOS</td>
<td>Distributed Denial of Service</td>
<td></td>
</tr>
<tr>
<td>QoE</td>
<td>Quality of Experience</td>
<td></td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
<td></td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
<td></td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
<td></td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td>Uniform Resource Locator</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>WinHTTP</strong></td>
<td>Microsoft Windows HTTP Services</td>
<td></td>
</tr>
<tr>
<td><strong>XML</strong></td>
<td>Extensible Markup Language</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The evolution of high wireless bandwidth and multimedia applications leaves no difference between mobile web and standard desktop environments. With the large screen sizes, even more than 3 inches smart phones have become an essential part of peoples colorful life. The increasing prevalence of smart phone use worldwide has come out with many questions on Quality-of-Service (QoE) and Quality-of-Experience (QoS). Today, web pages are getting bigger and complicated in order to meet modern requirements. The size of an average web page has increased almost 70% over the last fifteen years and average number of request handled by a browser has increased by factor 40% [2]. A recent research indicates that, mobile browsing will grow over 900% by 2014 [3].

Web latency is an important parameter, affecting the web page providers business currently. According to online shoppers survey, conducted in 2006: 75% customer will leave the websites if it takes more than 4 sec to render a page [4]. Over the years, numerous modifications in hardware, operating systems, browsers and physical network resulted in limited improvement in page loading times (PLTs). However, these enhancements seem to be device dependent. In case of mobile devices [2] with low processor speed and limited bandwidth, the efficiency is almost negligible. Due to hardware constraints, web pages load too slowly especially on mobile devices [5].

HTTP and TCP had not designed to reduce latency, therefore require some modifications according to current internet traffic demands. For this reason, a lot of research performed at application layer and network level with the goal to make the web faster.

This thesis mainly aims at the performance evaluation of new application layer protocols like SPDY that contribute to reduce PLT.

1.1 Motivation

The underlying protocol that contribute in the mobile web browsing are HTTP, TCP and Domain Name System (DNS). The vulnerability of DNS
introduces the slow update of DNS records, Distributed Denial of Service (DDoS) attacks and the most crucial, longer address resolving process. DNS resolution time is the main concern regarding mobile web browsing. Different approaches like alternate DNS services, enhancing DNS services and DNS prefetching adopted to address this issue. The reliable delivery of data makes TCP a perfect candidate for transporting web-browsing traffic. In the past few years, TCP has encountered various changes from its basic structure due to changed network conditions and parameters. Typically, TCP does not support multiplexed and multi-streamed connections, which is a major drawback. TCP handles each request either encapsulated in a separate connection or serialized over a single connection. The first causes slow start and 3-way-handshaking for each connection. The latter introduces the problem of head-of-line (HOL) blocking. Under any circumstances presented above, TCP experiences a performance loss.

HTTP is a popular subject when it comes to web traffic analysis. It offers a widely employed request-response model for web browsing. There remain other protocols for accessing internet information such as File Transfer Protocol (FTP), Secure Shell (SSH), Telnet, etc. HTTP is the only renowned protocol for web browsing and information retrieval. HTTP has advanced from HTTP/0.9 to HTTP/1.0 with many modifications in its basic structure. Numerous internet applications utilize HTTP protocol, as these require a clear communication port to pass through the strict firewalls, routers access lists and Network Address (NAT). Although, HTTP is preeminent and highly comparable to other application level protocols, yet it exhibits some shortcoming, which needs to be rectified.

A modern web resource consists of HTML document, Cascading Style Sheets (CSS), pictures, audios, videos flash and JavaScript files. Regardless of content attributes and design, domain sharding is a special feature of web content. According to the volume, web contents are distributed across multiple sub-domains to reduce the load on a particular server and to enhance the performance. A client can establish multiple TCP (Ntcp) connection with one web server. By splitting the content across multiple domains (Ndom) in this fashion, a client concurrently configures Ntcp x Ndom TCP connections and gets access to Ntcp x Ndom resources. DNS lookups increase per domain, which may increase or decrease PLT [6].

Web browsers are conceded as slow for taking many seconds to render a webpage. This lofty hold-up harms user perception and eventually damages the web-based business. Caching and pre-fetching are also ineffective [7] for speeding up mobile browsers. The first fails because many of the resources are not in cache and cached copies expire quickly. The second causes additional wireless network data usage with a little performance improvement. This is mainly due to average network conditions and limited mobile device resources [8]. Given the same content and network conditions, different devices show massive difference in the PLD [9]. Some recent research endeavors
motivate our choices. A recent study [9] illustrates that long network RTT determines the slow browser performance. Apart from all the improvements at the hardware and software side, bandwidth and RTT are the key factors for higher page latency. Indeed, RTT turned out to be more important than bandwidth. An increase in bandwidth from 5 Mbps to 10 Mbps only reduces the page load time by 5% [10]. Regardless of current bandwidth, a significant reduction in the RTT can make the web browsing faster. Considering all the facts presented above, it was essential to know whether the newly proposed protocols bring any improvement in mobile web performance or not.

HTTP optimization should also speedup mobile browsers and applications. Mobile devices exhibit distinct features in terms of processor speed, memory usage, limited battery life and connection cost in comparison with desktop computers. In order to achieve fast and cost efficient mobile web browsing, developers should consider all the factors like speed, memory, power and bandwidth at the same time for the optimization of a protocol.

### 1.2 Research Questions

The research questions addressed throughout this thesis are listed as follows:

1. How can we evaluate application layer web-latency protocols in mobile environments?
2. How do SSL handshakes impact the SPDY and HTTP performance?
3. What is the correlation between variable bandwidth and RTT in terms of speeding up mobile web browsing?

### 1.3 Research Methodology

We have followed Snowball literature Search for literature survey. It consists of a bunch of research papers, a master thesis [11], that is used as a reference for this study. Two books, [2] and [12] that laid the foundation of this thesis work and provided a very sound motivation to execute this research.

We have conducted a client-server based web browsing experiment. A dummy web page is prepared and distributed over seven domains on the server which is Virtual machine running over physical server. We have developed testing system to keep the whole testing processes automated. A python script is running on the client machine that controls the browser. The browser requests the page with the help of python program and records the request/response time stamps. The program calculates the difference between the two stamps and stores the result in MongoDB database.
Chapter 2

Related Works

There are several existing and forthcoming approaches to improve PLT for a given webpage. We summarize our literature search from the viewpoint of conventional approaches and some recently implemented application layer protocols to improve web performance. We also discuss some proposals that are still in the IETF drafts.

In [13], HTTP/1.1 offers persistent connection that enables the client to establish one TCP connection, through which the client can access several page resources from a particular web server. The client sends a GET request to a web server and receives the response. HTTPs persistent connections considerably reduce overhead evoked by establishment/termination of TCP connection and improve the web performance. In the study presented by Craig and Gregory [14], the authors use a single bundle object consisting of set of objects embedded on a web page for retrieval by the clients. They report that clients experience better response time when retrieve web pages as a single bundle of objects. Based on the data, they conclude that use of bundles exhibits reduced download times and lessens the average object delays as compared to HTTP/1.0 and HTTP/1.1. Moreover, this mechanism requires no change in the current HTTP protocol. The paper [15] focuses on the performance comparison of HTTP and SPDY. The authors investigate the effect on PLT by applying different mechanisms offered by the two protocols. They employ Network emulator and DummyNet to simulate variable bandwidth and RTT.

They describe that PLT is the main performance metric that is hard to define and measure correctly. The test bed consists of two machines and client is running with chromium browser. A simple C# program drives the browser. For the experiments, both dummy and actual web pages are stored in the browser cache to highlight the certain aspects of performance. WinHTTrack application arranges a mirrored copy of all actual websites in the browser and allows a user to browse it from link to link, hence provides a complete actual online view. They observe that SSL handshakes have a significant impact on SPDY performance. In addition, the features like pipelining and page minification can bring HTTPs performance close to SPDY. Unfortunately, authors
could not mention any correlation between variable RTT and bandwidth in their result representation.

A passive measurement approach is selected in [16]. The authors evaluate SPDY performance on a real browser and mobile phone by fetching 77 URLs from different websites. The test bed comprises of GFE (Google Front End) and Web Page Replay (WPR) server, ran on two separate Linux desktop machines. WPR server hosts all cached web contents, stored on its local hard disk to eliminate additional sources of latency. The GFE was configured as proxy to WPR server that actually hosts actual web contents. The measurements does not include real network DNS lookups delays and packet loss for these reasons; 1) IP addresses for all domain lookups are configured to return to GFE machines IP address, 2) real cellular networks hide packet loss at PHY layer. 3G network is emulated using DummyNet. For some reliability, authors took the metrics values (i.e. bandwidth and RTT) as a representative of cellular network performance in United States. They state that SPDY achieves 23% reduction in mean PLT over HTTP. Regardless of results, authors did not mention how many times they fetch a webpage to calculate mean PLT. Moreover, they did not consider SPDY header compression feature specifically in mobile networks measurements. Nonetheless, SPDY reduces header sizes by 85-88%.

In [17] authors have applied HTTP-pipelining, a method that allows a client to make multiple requests without waiting for each response, allowing a single TCP connection to be used much more efficiently. It is purely a client-based solution for downloading a single large file over multiple interfaces. In other words, by dividing a single large file into small-sized data packets can reduce the latency. Requesting the large amount of data introduces additional time overhead and reduces the throughput. HTTP pipelining eliminates the waiting time between successive requests and efficiently increases the throughput.

Deficiencies of HTTP and TCP resulted in some new approaches at transport and session layer level. These approaches turned out to be promising concerning efficiency and latency. HTTP transactions over a single TCP byte stream introduce HOL blocking in high latency and lossy browsing conditions and worsen the web response times. Multiple TCP connections may lower HTTP throughput and eventual performance depends upon the path latency, available bandwidth and HTTP response sizes and network congestion. TCP cannot logically separate HTTP data in its transport and delivery mechanisms. SCTP (Stream Control Transfer Protocol) reduces HOL blocking by offering streams with logical data flow and its own control sequences. A web client over TCP needs to decide the optimal number of TCP connections that reduce the HOL blocking with minimum impact on HTTP throughput. In comparison with this SCTP eliminates [18] HOL blocking by increasing number of streams while HTTP throughput remains unaffected simultaneously.
Chapter 3

Technical Background

Mobile web browsers are richer applications providing a flexible platform for extensive web interaction. The prominent layout engines such as Web kit [19], Presto [20], Gecko [21] and Trident [22] set the framework for modern mobile web browsers, similar to PC based browsers. Though the layout engine differs in many aspects, yet the browsing experience is same in all the engines.

3.1 Web Browsing in Cellular Networks

This sub-section presents a brief overview of mobile web browsing. Mobile web browsing comprises of several entities like radio channel allocation, DNS queries, SSL/TLS negotiation, TCP connection establishment, HTTP request and responses and resource fetching of browser. There exist different metrics that can be tuned to achieve maximum performance in mobile networks. We describe the steps that are involved in mobile browsing based on the assumption that the wireless connection is available and browser is idle and waiting for a request from the user.

**Step 0** User indicates the address of the desired website to the mobile browser using his/her favorite search engine and presses Enter/Go.

**Step 1** DNS resolver inspects for the desired information in DNS cache and sends the request to the DNS server to resolve the Domain Name to the network IP address, if the information is missing.

**Step 2** If the wireless network is not alive then the DNS query from the DNS resolver calls upon the wireless network in the mobile device, instigating the user to connect to the wireless network.

**Step 3** Once the mobile device connects to the wireless network, it is assigned a unique IP address by service provider. This public/private IP address may be IPv4 or IPv6 depending upon the availability and operators policy.

**Step 4** DNS requests sent to DNS server on the network result in DNS responses, which constitute IPv4 or IPv6 address(es).

**Step 5** The web browser grabs IP address(es) of the website, checks the existing TCP connections and tries to reuse the already established TCP
connection. In any of the two cases mobile browser, setup a TCP connection between mobile device and web server.

**Step 6** Every website follows a certain security policy. Based on the policy an established connection can be secure (https) or insecure (http). A browser introduces a certificate exchange with the server to validate the authenticity of the website.

**Step 7** After the TCP connection establishment and SSL/TLS handshakes the browser is ready to serve the user requests. HTTP request and responses begin as the user demands for a web page. Web server responds with all the resources in order to the requests sent from the web browser.

**Step 8** HTTP protocol retrieves all the object from the web server. Web browser then fetches all the objects from HTTP protocol and displays it on mobile screen. Web browser uses a specific lay out engine to presents all the objects on mobile device screen, as described earlier in this section.

**Step 9** The objects of a website are distributed over different domains called sub-domains, depending upon the size of the website. Each domain has its own unique IP address. In this regard, DNS resolution and step 6, 7, 8 and 9 are repeated, when fetching each object from different domain until the entire website is retrieved from the web server.
Figure 3.1: Message flow in Mobile Web Browsing

From the figure, we observe that user types the URL (www.facebook.com) in the address bar of the mobile browser and requests this web page. The user gets access to the wireless network through DNS resolver program in mobile device. DNS queries are forwarded to get IP address(es) against www.facebook.com. When DNS resolution provides IP address(es) mobile device establishes a TCP connection with the particular server. HTTP GET request is sent to render index.html. A mobile browser identifies images and CSS file present in the index.html. As CSS file resides on the same domain as index.html, a request is sent to retrieve the CSS file. At the same time,
browser sends another DNS query to resolve sub-domain address to retrieve images. Once obtained the IP address, browser establishes another TCP connection between the sub-domain and the mobile device. The user can view the complete website when browser retrieves all the objects.

3.2 HTTP Requests and Responses

HTTP follows a request-response model for communication between client and server. A client initiates the transaction by sending request to the server for a particular page; server generates the response with the page and resources associated to the page. Depending on the number of objects existing on the website, a browser may send multiple requests to the server. In practice browser, first sends a request for the index page with text and then initiates a subsequent request to retrieve image and CSS file. Once received, browser rearranges all the objects according to the website structure. The time needed to retrieve all the resources or the network latency is the ultimate delay in this model. In case of resource retrieval time, a bad cache control may be the reason for the delay. An efficient cache control can reduce load on a web server largely. Comprehensive HTTP payload compression methods can sufficiently reduce the effect of high network latency. There are several HTTP payload compression techniques however, both client/servers must agree on the selected compression method before starting the communication.
There exists a huge use of resources in the modern mobile web browsing. This extensive usage demands the optimization of the involved components to achieve maximum throughput. We are mainly concerned with the optimization of HTTP, alternative protocols and approaches to improve the web performance. The following sections describe the drawbacks in current protocols and proposed solution to address the problems.

4.1 Deficiencies in Current Protocols

The typical HTTP application in modern browser can only fetch one resource per single TCP connection. The pipelining is another feature in HTTP/1.1 [23]. It allows a client to send multiple requests over a single TCP connection. The server generates responses against these requests in order. Multiple requests over same TCP connection may give rise to HOL blocking. In fact, in a queue (FIFO) a delay in the response to the first request can block all the forthcoming responses. It is clear from the research that most of the modern browser and server do not avail pipelining feature of HTTP. It is relatively easy to implement pipelining on servers if we do not flush the network buffers, when pipelined requests are in queue. Web browsers like Safari [24] and Chrome [25] do not offer pipelining feature.

However, Mozilla Firefox [26] presents this feature, enabled by default and Opera 4.0 [22] supports it by default. In addition to pipelining, multiplexing feature is of great importance. HTTP/1.1 has chunked-transfer encoding, a similar feature to multiplexing. A server breaks down the data into small chunks when the length of the content is unknown. Although chunking allows the sender to split data, yet it does not ensure the interleaved content distribution of resources. Multiplexing helps server to establish persistent TCP connections, thus avoids the delay in establishing new TCP connections.

The number of TCP connection per domain is limited. This is a bottleneck as the numbers of concurrent requests on those TCP connections are also limited. In this case, TCPs slow start plays a major role in overall performance. During the slow start phase, the size of the initial window is initially
small. The size of the initial window increases with the number of TCP connections. It takes time for the TCP to achieve maximum throughput during the slow start phase.

This performance degradation can be overcome by establishing multiple TCP connections. A website contents are distributed over different domains called sub-domains. A web browser can make multiple requests over multiple TCP connections and client receives the objects in much faster way. Currently Opera 9 [20] can establish 2 to 6 or 8 parallel TCP connections per domain. Multiple TCP connections per domain are suitable for the high bandwidth networks with low latency.

A traditional HTTP server cannot push the objects towards the client until the client has sent request for it. This leads to both positive and negative effects. Server may push the resources that are not useful for the client. This will cause an inefficient use of bandwidth resulting in additional latency. A supposition may be that client requires some information from the server like meta-data and navigation instructions through the website it visits for the first time. In this case, client will desire some suggestion and hint from the server. A similar feature to the server push is request prioritization. HTTP client is not allowed to prioritize the request or demand server to send essential objects to render the website. The server-push and request prioritization feature holds a key importance in mobile environments, where the bandwidth is limited.

Header and data compression have been the part of debate over the years. We should critically analyze the gain in compressing HTTP headers. HTTP headers are of few KBs in size in comparison with actual payload. Nonetheless, HTTP header compression is only beneficial in minimal bandwidth networks. In addition, removing the redundant data in HTTP headers can be another way to improve the performance in high latency networks.

HTTP/1.1 employs content encoding and data encoding for data compression. PLT in HTTP has been a primary concerns for developers and designers over the years. The improvement solution include: a) increasing the number of connections per domain, b) optimizing the resource size, and 3) distributing contents over multiple domains.

### 4.2 Proposals for Faster Web

This section focuses on some approaches for HTTP optimization and proposal for a new protocol SPDY for faster transport of content over web.

#### 4.2.1 Header Compression

All modern browser support compression feature but there are still many cases where users suffer from receiving uncompressed data. According the
experiments conducted by Google only two third of the actually compressible content on the web is actually compressed [27]. This causes an inefficient usage of bandwidth and minimizes the quick responses from the server. In resource and bandwidth limited devices it takes additional round trips to transfer the pages due to uncompressed data, which hurts end user. However, we are mainly concerned with the header compression, which is 200 bytes to 200 KB. Using header compression both request and response size can be minimized.

Uncompressed headers cause additional overhead on both client and server side. Moreover, redundancy in the subsequent client/servers can be avoided after initial negotiation. This leaves the overhead mainly on the client and server side. The client and server should agree on a particular method for successive header compression. Header compression increases the processing load on the end devices but reduces the data on wire by sending compressed, small packets

4.2.2 Request Prioritization

Request prioritization allows a client to send request for as many items it wants from the server and assign the priority to each request. For instance, if client requests for a particular object and browser renders that object at the end. This will result in a negative user perception. When browsing Gmail, a client instantly wants login information box to appear. When login box appears in the end and images logos and other text appears first, it would result in unhappy end user experience. It is interesting to consider if client can assign a priority for the login box and server can push it first.

The situation in which objects are present on different sub-domains, the primary server will be unable to transfer or push the required object. The client can retrieve the most important object using meta-data. The request prioritizing is possible by pre-loading meta-data of the website. The request prioritization is advantageous when a typical user wants to navigate through the website before it is fully loaded. Browsers tend to develop DOM (Document Object Model) tree before creating a render tree. Generally, browsers display the objects on screen at the time they are fetched from server and create the render tree later. The request prioritizing is utilized to speed up this process.

4.2.3 Multiplexing

HTTP/1.1 has provision for pipelining but does not support multiplexing. We assume that the multiplexing is available in HTTP, then it can send multiple objects interleaved in a single TCP connection. A client establishes a TCP connection and initiates a request to the server to render particular object. If the TCP connection is already open, HTTP can make use of its
persistent connection feature and requests for the particular object. Persistent connections are suitable when client needs to fetch several objects from the server over a short time period.

Considering the case where client has already opened TCP connection and requires additional objects from the server. Client either can use the existing TCP connection or may establish a new TCP connection. This leads us to two possibilities: 1) client is in the middle of receiving object from the server, 2) TCP connection is idle, and there is no data flow between the two ends. For the first case client can use multiplexing and avoids the additional delay for establishing new TCP connection. If the connection is no active, there is no need for multiplexing, as there is only one object to be transferred. Multiplexing effectively clears up the problem of HOL blocking in HTTP pipelining. The available object can be transferred immediately to the network without waiting for the blocked response using multiplexing.

**Figure 4.1:** HTTP Pipelining
Figure 4.2: Multiplexing
Chapter 5

Methodology

In this section we have elaborated the test setup to measure the performance of HTTP, HTTP-Pipelining, and SPDY. The goal was to analyze the impact of these latency-reducing protocols on PLT by varying bandwidth and RTT. We have developed an automated framework that can measure the webpage load time under variable network conditions.

Figure 5.1: Basic Experimental System

Figure 5.1 illustrates the three main components of experiment system and their connections. This chapter will provide detail description about the experimental process, including the tools and techniques, which were deployed to perform tests.

5.1 Testbed and Setup

The test bed consists of client running Google Chrome on a physical machine that supports all modern protocols. Server is deployed on the same physical machine using virtual box. Server is configured to support all latency-reducing protocols. The traffic is generated using DummyNet to simulate different network conditions. A shell script is running on client to setup the apache web server and DummyNet. A python script is written to invoke the web browser. It employs selenium framework to control the web browser.
Chapter 5. Methodology

5.2 Network Setup

We used DummyNet to simulate different kinds of network condition. It is easy to install and control. DummyNet is deployed at server side. All the HTTP and HTTPS traffic was shaped. In order to simulate enormous variable network conditions, its impossible to stop the test and modify the network parameters to DummyNet; we wrote an automated script to change the values of bandwidth and RTT for every test case. Before a new test, the script will setup DummyNet with certain parameters on the server side using SSH technique. We synchronized our Linux system with NTP server using /etc/ntp.conf file for measuring accurate time stamps. We have used TCP under default settings on Linux system.

5.3 Automation Testing Framework

In order to perform several test cases we have chosen Selenium, a framework that invokes web browser to perform relevant tests automatically. WebDriver is a protocol which provides a programmable interface to the web browsers. This allows developers to write some programs to finish the automation testing jobs come true. Figure 5.3 shows the normal work flow of Selenium and WebDriver We wrote a Python script using Selenium WebDriver library. This script now can control the web browser to request the dummy webpage from the server. The internal communication between script and web browser uses WebDrvier protocol. However, the communication between web browser and web server is purely through HTTP protocol. The page load time will be recorded in the script and store the results in the database finally. Figure 5.2 shows the flow of our automation testing framework.

5.4 Client Side Configuration

The client is running Google Chrome as web browser hosted on Linux based system with Ubuntu distribution 12.04. MongoDB is used as a database to store the results. MongoDB is a NO-SQL database which uses JSON as its exchange data format. It looks very close to data structures of many programming languages. The domains used in the dummy page were real, so we had to map the domain names to their own IP addresses on the server. Its done by editing file /etc/hosts.
5.5 Server Side Configuration

Since we only had one computer, therefore we utilized Virtualization software to have more virtual servers. We used Virtualbox. The server operating system was Ubuntu 12.04 server 32-bit. HTTP server was Apache 2.2 which supports HTTP and HTTP Pipe-lining natively. In order to support SPDY, we installed mod-spdy, a SPDY module for Apache. Dummynet was also installed as network shaper. The dummy web page is distributed over 7 domains. Table 5.5 lists name and the corresponding IP addresses of each domain.

Table 5.5: Domain Names and Distributed Contents

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Hosted Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>en.m.wikipedia.org</td>
<td>main HTML file</td>
</tr>
<tr>
<td>bits.wikimedia.org</td>
<td>icon files</td>
</tr>
<tr>
<td>bits-02.wikimedia.org</td>
<td>JavaScript files</td>
</tr>
<tr>
<td>en.m.wikipedia.org</td>
<td>CSS files</td>
</tr>
<tr>
<td>upload.wikimedia.org</td>
<td>big picture files</td>
</tr>
<tr>
<td>upload-01.wikimedia.org</td>
<td>small picture files</td>
</tr>
<tr>
<td>upload-02.wikimedia.org</td>
<td>thumbnail picture files</td>
</tr>
</tbody>
</table>

5.6 Hardware Configuration of Equipment

Table 5.6 provides the detail hardware configuration of the machines that were used in the experiment.

Table 5.6: Hardware Configuration

<table>
<thead>
<tr>
<th>Host</th>
<th>Hardware</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>Intel ci7 CPU 930 @ 2.80GHz</td>
<td>Ubuntu server12.04 x86</td>
</tr>
<tr>
<td>Client</td>
<td>Intel ci7 CPU 930 @ 2.80GHz</td>
<td>Ubuntu server12.04 x86</td>
</tr>
</tbody>
</table>

5.7 Software Configuration of Equipment

The following tables provide the detail software configuration of the machines that were used in the experiment.
Table 5.7: Server side Software Configuration

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>2.2.22</td>
<td>HTTP Server</td>
</tr>
<tr>
<td>mod-spdy</td>
<td>0.9.3.3-r386</td>
<td>SPDY module for Apache</td>
</tr>
<tr>
<td>DummyNet</td>
<td>20120812</td>
<td>Network Emulator</td>
</tr>
</tbody>
</table>

Table 5.7.1: Client side Software Configuration

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu</td>
<td>12.04 server x86 64</td>
<td>Host OS</td>
</tr>
<tr>
<td>Virtual Box</td>
<td>4.2.10-84104</td>
<td>Virtualization Software</td>
</tr>
<tr>
<td>Gogle Chrome</td>
<td>26.0.1410.63-r192696</td>
<td>Web browser</td>
</tr>
<tr>
<td>Selenium</td>
<td>2.30.0</td>
<td>Software Testing Framework</td>
</tr>
<tr>
<td>Chrome Driver</td>
<td>26.0.1383.0</td>
<td>WebDriver for Chrome</td>
</tr>
<tr>
<td>Vagrant</td>
<td>1.0.7</td>
<td>virtual development environments</td>
</tr>
<tr>
<td>Mongo DB</td>
<td>2.2.3</td>
<td>Programmer friendly Database</td>
</tr>
</tbody>
</table>

5.8 Dummy Webpage

The table 5.8 provides the detailed composition of Dummy webpage.

5.9 Testing Methodology

The work flow of the experiments is controlled by python and shell scripts running on client and server end respectively. After booting to system, using Vagrant to start up the server in virtual machine. There were a few things needed to check on the server: DummyNet kernel module, IP addresses of domains and Apache service.

Step 1. Startup server in the Virtual Machine

Step 2. Execute main controller script newrun.sh with a test case configuration file as the parameter. e.g. ./newrun.sh spdy-non-ssl.case.

Step 3. newrun.sh imports a few variables from the test case configuration file which tells the settings of tested protocol and network.

Step 4. newrun.sh calls setup Apache function to invoke Puppet script on the server to make Apache enable one certain protocol by using the protocol and SSL variables from step 2.

Step 5. newrun.sh calls setup DummyNet function to invoke setnet.sh script on the server to make desired network take effect by using the RTT,
Table 5.8: Dummy Webpage Composition

<table>
<thead>
<tr>
<th>Elements</th>
<th>Amount</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>36</td>
<td>360 KB</td>
</tr>
<tr>
<td>Domains</td>
<td>7</td>
<td>/</td>
</tr>
<tr>
<td>HTML File</td>
<td>1</td>
<td>87 KB</td>
</tr>
<tr>
<td>JavaScript File</td>
<td>2</td>
<td>50 KB</td>
</tr>
<tr>
<td>CSS File</td>
<td>2</td>
<td>17 KB</td>
</tr>
<tr>
<td>Pictures</td>
<td>31</td>
<td>213 KB</td>
</tr>
</tbody>
</table>

download bandwidth values and LOSS variables from step 2.

Step 6. newrun.sh calls run.py to invoke Google Chrome web browser to request the dummy web page prepared on the server. It calculates the total load time of each request and save into database.

Step 7. The steps 5 and 6 repeats with a different network condition combination (download bandwidth and RTT).

Figure 5.3 show that browser is controlled by client side program, written in python. Client request for web page from the server and stores the time stamps in data base.
Chapter 6

Analysis and Result

For the experiments we have used a Dummy web page comprises of text and images java scripts and CSS. The total size of web page is approximately 320 KB [29] as per Google recommended web metrics. RTT is the main factor that plays a vital role in improving PLTs. We simulate variety of RTTs and bandwidths to investigate the performance comparison of HTTP, HTTP-pipelining, and SPDY.

6.1 Page Load Time Calculation

Page load time (PLT) is calculated through script 'run.py'. It records the current timestamp as browser opens and request is sent to fetch the dummy webpage. The final time stamp is recorded when the browser finishes request and closes. Hence, PLT is the difference of the two calculated times tamps. We can physically observe the opening and closing of the chrome browser on machine.

6.1.1 Data Selection

We have selected the best data set out of five data sets on the basis of standard deviation and calculated 95% CI. We are presenting one data set as we have a long range of variables for RTT and bandwidth.

6.1.2 Calculation of Mean and Standard Deviation

The Mean, Standard Deviation and Confidence Interval for the Page Load Times are calculated for each value of bandwidth and RTT.

Let,

\[ X \] be the Page Load Time for each request of the dummy webpage
Chapter 6. Analysis and Result

The Mean Page Load Time is defined as,

$$\bar{X} = \frac{1}{N} \sum_{i=1}^{N} X_i$$

Where $N = 5$, $X_i$ = Page Load Time for the $i^{th}$ request.

The Standard Deviation of the Page Load Time for first web page request is defined as,

$$\delta_p = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (X_i - \bar{X})^2}$$

6.1.3 Calculation of Confidence Interval

Once all the results of Mean Page Load Time were calculated. The Confidence Intervals (CI) for all the Mean Page Load Time were calculated as the mean is always related with CI, 95%. The fitness of the calculated Mean of the Page Load Time can be obtained using 95% CI.

The 95% of CI for Page Load Time of any session is defined as,

$$[\bar{X} - \gamma_p, \bar{X} + \gamma_p]$$

Where marginal error $\gamma_p$ is defined as,

$$\gamma_p = 1.96 \times \frac{\delta_p}{\sqrt{N}}$$

$\delta_p = $ Standard Deviation of Page Load Time for the first request
$N = $ Number of requests by the client browser

6.2 HTTP and SPDY Results SSL

Scenario#1: Vary the Bandwidth
We have run this test to observe the PLTs offered by HTTP and SPDY. We have introduced a standard packet loss of 3% [1]. This packet loss is divided equally for the uplink and downlink traffic. We have set a various
fixed value for RTT and bandwidth is varied. HTTP and SPDY are running under SSL and Non-SSL configuration, which leads us to a fair comparison between these protocols. We have fetched Dummy page five times for each value of bandwidth from seven different sub-domains. We have calculated mean, standard deviation and 95% CI that are given in the appendix C. The mean PLT is presented in the graph. Here are the results using HTTP, HTTP-pipelining, and SPDY:

\[ \text{Figure 6.1: Impact of SSL Handshake Delays on SPDY Performance} \]
Chapter 6. Analysis and Result

Figure 6.2: Impact of SSL Handshake Delays on HTTP Performance

It is clear that for a variable bandwidth both SPDY and HTTP exhibit a significant difference in PLTs when executed under SSL and non-SSL configuration. The same trend can be noticed if we keep the download bandwidth constant and vary RTT.
Scenario #2: Vary the RTT
In the next case we have varied RTTs for several fixed values of bandwidth.

Figure 6.3: SPDY Mode SSL, Download Bandwidth fixed at 10 Mbps

Figure 6.4: HTTP Mode SSL, Download Bandwidth fixed at 10 Mbps
6.3 HTTP, SPDY and HTTP-Pipelining Results

SSL

SSL is a cryptographic protocol that adds extra security to the data transfer. The SSL handshakes mechanism between client and server cause more delay when fetching a web page.

Scenario#3: Vary the Bandwidth

![Figure 6.5: Impact of SSL Handshakes on PLT](image)

We see that at low RTT of 150 ms HTTP consistently performs better than SPDY and Pipelining. It is important to observe that for a range of bandwidth from 3 Mbps to 15 Mbps the page load times for all three protocols are displaying the repeated trends. Therefore increasing the bandwidth is not the solution to reduce the page load times.
Scenario 4: Vary the RTT

It can be observed from the graph that SPDY tries to overcome HTTP performance with reduced RTT’s. In addition, it’s very much clear that RTT is the only factor that impacts web performance mainly. PLTs with SSL handshakes are minimally higher in comparison with non-SSL configuration.
We have evaluated the performance of HTTP, SPDY, and HTTP-pipelining. SPDY significantly tends to reduce the page load time in desktop environment where the bandwidth is high and hardware resources are not constrained. In high bandwidth networks bottleneck is minimal. However, in low bandwidth networks like mobile networks the concurrent TCP connection offered by SPDY are more important. In mobile networks low bandwidth for each TCP connection may lead to bottleneck in the network and negatively impact the SPDY performance.

We have demonstrated that adding extra bandwidth to a link results in a minimal reduction in page load time. However, decreasing the RTT leads us to better results. Moreover, reduction in TCP and DNS handshakes will also considerably improve the web performance. Therefore a modification is required in these protocols.
Chapter 8

Conclusions

This study mainly focuses on evaluation of latency reducing protocols in mobile environments. Our initial finding indicates that HTTP performs better than SPDY and HTTP-Pipelining in low bandwidth networks. Firstly, SPDY could not provide high speed web access due to multiple parallel SPDY sessions in mobile environments, since it takes multiple second to establish a SPDY session. Secondly, increasing the bandwidth is not the only solution to improve web latency. We have tested our results with broad range of bandwidth and RTT using smaller step size to verify the results. The results testify that RTT is the most prominent factor that directly affects the web performance in low and high link bandwidth networks. The data shows that SSL handshakes have a significant impact much on SPDY and HTTP performance. It is demonstrated that HTTP and SPDY performance is very close to each other in high bandwidth networks. Currently, it is not proved that SPDY should replace HTTP. More experiments are needed in a real network environment to draw a clear conclusion. We are unable to explain big spikes in the page load times of SPDY. We are currently investigating it further.

8.1 Answer to the Research Questions

The answer to the research questions are listed below.

Question 1. How can we evaluate application layer-latency reducing protocols in mobile environments?

Answer 1. We have presented an automation testing frame work to evaluate the performance of these application layer web latency reducing protocols. The setup consists of a client server model. A python program controls the client browser with the help of selenium WebDriver library. The program initiates the client request and record the time stamps of the interval during the request are sent and response is received. The results are stored in the database.
Question 2. How do SSL handshakes impact the SPDY and HTTP performance?

Answer 2. It can be seen from the graphs that SSL handshakes have significant impact on SPDY and HTTP performance. The PLTs are high when using SPDY and HTTP with SSL as compared to non-SSL mode.

Question 3. What is the correlation between bandwidth and RTT in speeding up mobile web browsing?

Answer 3. We have demonstrated that in low bandwidth and high bandwidth networks RTT is the only parameter that improves the speed of web transactions. The RTT delays are same even we double the bandwidth of a link.

8.2 Future Work

Through this research we have observed that bandwidth and RTT play an important role in determining user perceived latency. Nevertheless RTT turned out to be the decisive factor in improving the web latency. We have neglected the RTT for DNS lookups since we have performed the experiment in controlled scenario where client and server reside on a same physical machine. In future we would like to perform the experiments on real network. We would like to include the impact of DNS lookups and TCP handshakes to evaluate the performance of these protocols. In addition, we would also like to evaluate the SPDY and HTTP performance with increased TCP frame size.
APPENDIX A

Client side Script
Online code repository address: https://github.com/vvoody/test_spdy

newrun.sh - main shell script to control everything

#!/bin/bash

# Run a complete round test of 1 protocol under all network combinations,
# which comprise all RTT and DW_BW.
# SSL and non-ssl are two separate rounds.
# # use 'DEBUG=echo bash $0' to do a dry run.
#
# $1 is the test case config file.
if [ $# -ne 1 ]; then
    echo "Usage: $0 test_case.conf"
    exit 1
fi

if [ -r $1 ]; then
    source $1
else
    echo "File '$1' does not exist."
    exit 1
fi

LOG_FILE=run.log
RUN_TIMES=5
# 'vagrant' host should be existed in ~/.ssh/config
REMOTE=${REMOTE:-vagrant}
# PROTOCOL should be lowercase, otherwise puppet cannot find the class.
# but run.py doesn't have to.
PROTO=${PROTO/-/}
# DEBUG=echo bash $0
DEBUG=${DEBUG:-}

URL='en.m.wikipedia.org/

if [ $SSL == "true" ]; then
    SSL_OPT="-s"
else [ $SSL == "false" ]; then
    SSL_OPT=""
else
    echo "'SSL' of case file was set incorrectly, which is '$SSL'."
    exit 1
fi

function setup_apache()
{
    fab -H $REMOTE set_apache:proto=$1,ssl_enabled=$2
}
function setup_remote_network() {
    fab -H $REMOTE set_net:args_str="$1",var=$2
}

$DEBUG fab -H $REMOTE touch_dummy_apache_config_files 2>&1 | tee -a $LOG_FILE
$DEBUG setup_apache ${PROTO,,} $SSL 2>&1 | tee -a $LOG_FILE

#echo "Please input something to continue... 
#read user_input 
#echo User_input

for dw_bw in $DW_BW_VALUES; do
    DW_BW=$dw_bw
    UP_BW=$(echo "$DW_BW/2" | bc)
    ARGS_STR="-l $LOSS -d $DW_BW -u $UP_BW -r"

    for rtt in $RTT_VALUES; do
        echo "**********" $(date) "**********" >> $LOG_FILE
        (cat <<EOF
        ============================================
        ===============
        =================
        Starting test of protocol '$PROTOCOL' with SSL($SSL) and parameters:
        $ARGS_STR $rtt
        under '$NETWORK' network.
        =========================================================================
        ===============
        EOF
        ) | tee -a $LOG_FILE
        $DEBUG setup_remote_network "$ARGS_STR" $rtt 2>&1 | tee -a $LOG_FILE
        $DEBUG sleep 1
        $DEBUG fab -H $REMOTE get_net 2>&1 | tee -a $LOG_FILE
        $DEBUG sleep 1
        $DEBUG python -u run.py -p $[PROTOCOL,,] $SSL_OPT -n $NETWORK $ARGS_STR $rtt $RUN_TIMES -v 2>&1 | tee -a $LOG_FILE
        # first time will fail for unknown reason, have to try twice.
        $DEBUG fab -H $REMOTE reset_net 2>&1 | tee -a $LOG_FILE
        $DEBUG sleep 1
        (cat <<EOF
        END of this round test.
        =========================================================================
        EOF
        ) | tee -a $LOG_FILE
    done
done

$DEBUG fab -H $REMOTE reset_net 2>&1 | tee -a $LOG_FILE
# END.
run.py - Python script which control Chrome and save results to
database via Selenium lib

---

```python
import argparse
import time
import json
from datetime import datetime
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.chrome.service import Service

# Chrome + Selenium testing script
# This script won't setup the network condition, the given arguments
# will be used to store to database together with test results.

class TestCase():
    def __init__(self):
        self.parser = argparse.ArgumentParser()
        self.parse_arguments()

        self.load_conf()

        self.chrome_driver = Service(self.CHROME_DRIVER_PATH)
        self.results = []

        # some kind of ID of a test
        self.when = int(datetime.utcnow().strftime("%s"))

        self.protocol = self.args.protocol
        self.chrome_options = self.decide_chrome_startup_options()
        self.ssl = self.args.ssl
        self.net_type = self.args.net_type
        self.net_up_bw = self.args.net_up_bw
        self.net_dw_bw = self.args.net_dw_bw
        self.net_rtt = self.args.net_rtt
        self.net_loss = self.args.net_loss

        self.request_url = ('https' if self.ssl else 'http') + '://' +
        self.REQUEST_PATH

        if self.args.verbosity:
            print self.args
            print self.chrome_options
            print self.conf

    def load_conf(self):
        with open('config.json', 'r') as f:
            config = json.load(f)
            self.conf = config

            self.CHROME_DRIVER_PATH = config.get('chrome_driver_path')
            self.REQUEST_PATH = config.get('request_path')
            self.DB = config.get('db')
            self.COLLECTION = config.get('collection')
            self.DBUSER = config.get('dbuser')
```

---
self.DBPASSWD = config.get('dbpasswd')
self.DBHOST = config.get('dbhost')

def decide_chrome_startup_options(self):
    opts = Options()
    opts.add_argument('--disable-extensions')

    if self.protocol == 'http':
        # server should be 'SpdyEnabled off'
        pass
    elif self.protocol == 'spdy':
        # server should be 'SpdyEnabled off' and
        # Chrome supports SPDY by default
        if self.args.ssl is False:
            # server: SpdyDebugUseSpdyForNonSslConnections 2
            opts.add_argument('--use-spdy=no-ssl')
    elif self.protocol == 'http-pipelining':
        # server should be 'SpdyEnabled off'
        opts.add_argument('--enable-http-pipelining')
    else:
        print "Won't reach here."

    return opts.to_capabilities()

def parse_arguments(self):
    self.parser.add_argument('-t', '--times', help='how many times to run',
                              type=int, default=10)
    self.parser.add_argument('-p', '--protocol', help='protocol name to test',
                              choices=['http', 'spdy', 'http-pipelining'], required=True)
    self.parser.add_argument('-s', '--ssl', help='SSL enabled',
                              action='store_true')
    self.parser.add_argument('-n', '--net-type', help='network condition',
                              choices=['EDGE', '3G', '4G', 'ALL'], required=True)
    self.parser.add_argument('-l', '--net-loss', help='packet loss ratio of
                              network(percent)', required=True)
    self.parser.add_argument('-u', '--net-up-bw', help='uplink bandwidth of
                              network(kbit/s)', required=True)
    self.parser.add_argument('-d', '--net-dw-bw', help='downlink
                              bandwidth of network(kbit/s)', required=True)
    self.parser.add_argument('-r', '--net-rtt', help='RTT of
                              network(milliseconds)', required=True)
    self.parser.add_argument('-v', '--verbosity', help='more debug info',
                              action='store_true')
    self.parser.add_argument('-x', '--dont-save', help='more debug info',
                              action='store_true')
    self.parser.add_argument('-z', '--holdon', help='seconds to hold on after
                              page loaded', type=int, default=1)
    self.args = self.parser.parse_args()

def run(self):
    self.chrome_driver.start()
    time.sleep(1)
    print "Test started at UTC time: %d." % self.when

    for i in range(self.args.times):
        driver = webdriver.Remote(self.chrome_driver.service_url,
                                self.chrome_options)
        time.sleep(2)
```python
print "No. of test: %d." % i
print self.request_url
start = datetime.utcnow()
driver.get(self.request_url)
end = datetime.utcnow()
t = (end - start).total_seconds()
print "Load time: %f" % t
self.results.append(t)

time.sleep(self.args.holdon)
driver.quit()
time.sleep(1)

def stop(self):
    self.chrome_driver.stop()

def get_mongodb_uri(self):
    host = 'localhost' if self.DBHOST is None else self.DBHOST
    user_passwd_str = "" if self.DBUSER is None else '{0}:{1}@'.format(self.DBUSER, self.DBPASSWD)
    uri = "mongodb://{0}{1}/{2}".format(user_passwd_str, host, self.DB)
    return uri

def dump_results_to_file(self, ds):
    with open('results.json', 'a') as f:
        json.dump(ds, f)
        f.write('n')

def save(self):
    import pymongo

ds = {'when': self.when,
    'protocol': self.protocol,
    'ssl': self.ssl,
    'net_type': self.net_type,
    'net_up_bw': self.net_up_bw,
    'net_dw_bw': self.net_dw_bw,
    'net_rtt': self.net_rtt,
    'net_loss': self.net_loss,
    'results': self.results,
    'load_time': sum(self.results) / len(self.results),
    'deviation': 0,
    }

try:
    conn = pymongo.Connection(self.get_mongodb_uri(), safe=True)
    db = conn[self.DB]
    collection = db[self.COLLECTION]
    collection.insert(ds)
except Exception as e:
    print "ERROR: %s " % str(e)
    self.dump_results_to_file(ds)
else:
    print "Test data saved to database."
    conn.close()

def main():
    # try here to close chromedriver server
    test = TestCase()
```
try:
    test.run()
except Exception as e:
    print "ERROR({0}): {1}".format(e.errno, e.strerror)
    print "chromedriver to be stopped."
finally:
    test.stop()

if test.args.dont_save:
    print "Test data not saved to database."
else:
    test.save()

if __name__ == '__main__':
    main()
APPENDIX B

Server Side Scripts:

setnet.sh - Shell script to setup DummyNet to simulate certain network

#!/bin/bash

usage() {
cat <<EOF
A simple wrapper of ipfw to configure Bandwidth, Delay and Packet Loss Rate of network.

Usage: $0 -b XXX [-d XXX] [-u XXX] -r XXX [-l X]

-h: Help
-b: Bandwidth (Kbit/s, uplink & downlink)
-d: Downlink Bandwidth
-u: Uplink Bandwidth
-r: Delay/RTT (ms)
-l: Packet Loss Rate (0.0~1.0, default to 0)

If you set the Bandwidth of target network with only `-b' option, then your network will be symmetric, which means bandwidth of both uplink and downlink is set to the value of the argument of `-b' option.

If you want an asymmetric network, you can use `-d' or `-u' option to set different bandwidth for downlink or uplink respectively. When one of them is used, it will override the value set by the `-b' option. Otherwise, use the value of `-b' option for the other link.

NOTE: This script requires `bc' command.
EOF
}

# if invoke this script like 'DEBUG=echo $0 -b 1024 -r 1000',
# nothing will happen.
DEBUG=$[DEBUG:- ]

if [ $# -eq 0 ];then
    usage
    exit 0
fi

while getopts "hfb:du:r:l:" opt; do
    case $opt in
        h)
            usage
            ;;
        f)
            ipfw pipe flush -f
            ipfw flush -f
            echo "Flushed ipfw done."
            exit 0
            ;;
        ...
b) 
BW_ALL="$OPTARG"
;;

d) 
BW_DW="$OPTARG"
;;
u) 
BW_UP="$OPTARG"
;;

r) 
RTT="$OPTARG"
;;
l) 
LOSS="$OPTARG"
;;

usage 
exit 1
;;
esac

done

if [ -z $BW_DW ] || [ -z $BW_UP ]; then
    if [ -z $BW_ALL ]; then
        echo "Please set bandwidth with at least '-b' option."
        exit 1
    fi
fi

BW_DW=${BW_DW:-$BW_ALL}
BW_UP=${BW_UP:-$BW_ALL}

if [ -z $RTT ]; then
    echo "Please set Delay/RTT with '-r' option."
    exit 1
fi

# Packet Loss Rate default to 0
if [ -z $LOSS ];then
    LOSS=0
fi

SDEBUG echo "Flushing previous rules..."
SDEBUG ipfw pipe flush -f
SDEBUG ipfw flush -f
SDEBUG echo "Flush done."

SDEBUG ipfw add 1 pipe 1 ip from any to any http,https inlet # uplink
SDEBUG ipfw add 2 pipe 2 ip from any http,https to any out # downlink

# network condition may be not symmetric
# bw: bandwidth, delay: RTT, plr: packet loss
SDEBUG ipfw pipe 1 config bw "$BW_UP"kbit/s delay $( echo "$RTT/2" | bc)ms
plr $( echo "scale=3; $LOSS/2" | bc)
SDEBUG ipfw pipe 2 config bw "$BW_DW"kbit/s delay $( echo "$RTT/2" | bc)ms
plr $( echo "scale=3; $LOSS/2" | bc)
http.pp - Puppet script to setup HTTP protocol support of Apache
==========================================================================

# mod_ssl is always enabled.
# HTTP & HTTP-PIPELINING only need to disable mod_spdy.

class apache::testcase::http($ssl_enabled='false') {
    include apache

    file '/etc/apache2/mods-enabled/spdy.conf':
        ensure => absent,
        notify => Service['apache2'],
    }

    file '/etc/apache2/mods-enabled/spdy.load':
        ensure => absent,
        notify => Service['apache2'],
    }
}

http-pipelining.pp - Puppet script to setup HTTP Pipelining protocol support of Apache
==========================================================================

# mod_ssl is always enabled.
# HTTP & HTTP-PIPELINING only need to disable mod_spdy.

class apache::testcase::http_pipelining($ssl_enabled='false') {
    include apache

    file '/etc/apache2/mods-enabled/spdy.conf':
        ensure => absent,
        notify => Service['apache2'],
    }

    file '/etc/apache2/mods-enabled/spdy.load':
        ensure => absent,
        notify => Service['apache2'],
    }
}

spdy.pp - Puppet script to setup SPDY protocol support of Apache
==========================================================================

# mod_ssl is always enabled.
# mods-available/spdy.conf__________ (ssl enabled)
# \
# ----> mods-enabled/spdy.conf
# / 
# mods-available/spdy-no-ssl.conf___/ (ssl disabled)
# | V 
# SpdyDebugUseSpdyForNonSslConnections 2

class apache::testcase::spdy($ssl_enabled='true') {
    include apache

    file '/etc/apache2/mods-enabled/spdy.load':
        ensure => link,
        target => '/etc/apache2/mods-available/spdy.load',
    }
notify => Service['apache2'],
}

if $ssl_enabled == "false" {
  file {'spdy-no-ssl':
    path => '/etc/apache2/mods-enabled/spdy.conf',
    ensure => link,
    target => '/etc/apache2/mods-available/spdy-no-ssl.conf',
    notify => Service['apache2'],
  }
} else {
  file {'spdy-ssl':
    path => '/etc/apache2/mods-enabled/spdy.conf',
    ensure => link,
    target => '/etc/apache2/mods-available/spdy.conf',
    notify => Service['apache2'],
  }
}
APPENDIX C

*Table C.1:* SPDY mode SSL with Bandwidth fixed at 10 Mbps

<table>
<thead>
<tr>
<th>RTT</th>
<th>Mean PLT[sec]</th>
<th>St.dev</th>
<th>Rel. 95% CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>3.62</td>
<td>0.20</td>
<td>4.91%</td>
</tr>
<tr>
<td>100</td>
<td>4.80</td>
<td>0.51</td>
<td>9.27%</td>
</tr>
<tr>
<td>150</td>
<td>5.50</td>
<td>0.51</td>
<td>8.13%</td>
</tr>
<tr>
<td>200</td>
<td>6.07</td>
<td>0.34</td>
<td>4.88%</td>
</tr>
<tr>
<td>250</td>
<td>8.38</td>
<td>1.66</td>
<td>17.32%</td>
</tr>
<tr>
<td>300</td>
<td>8.80</td>
<td>1.18</td>
<td>11.79%</td>
</tr>
<tr>
<td>350</td>
<td>9.42</td>
<td>0.88</td>
<td>8.21%</td>
</tr>
<tr>
<td>400</td>
<td>9.83</td>
<td>0.86</td>
<td>7.71%</td>
</tr>
<tr>
<td>450</td>
<td>11.56</td>
<td>1.59</td>
<td>12.07%</td>
</tr>
<tr>
<td>500</td>
<td>12.96</td>
<td>1.61</td>
<td>10.89%</td>
</tr>
<tr>
<td>550</td>
<td>15.25</td>
<td>0.60</td>
<td>3.43%</td>
</tr>
<tr>
<td>600</td>
<td>16.87</td>
<td>2.38</td>
<td>12.36%</td>
</tr>
<tr>
<td>650</td>
<td>15.27</td>
<td>1.48</td>
<td>8.49%</td>
</tr>
<tr>
<td>700</td>
<td>15.70</td>
<td>1.82</td>
<td>10.14%</td>
</tr>
<tr>
<td>750</td>
<td>18.67</td>
<td>2.34</td>
<td>10.97%</td>
</tr>
<tr>
<td>800</td>
<td>17.81</td>
<td>0.99</td>
<td>4.88%</td>
</tr>
<tr>
<td>850</td>
<td>19.32</td>
<td>1.77</td>
<td>8.05%</td>
</tr>
<tr>
<td>900</td>
<td>21.15</td>
<td>2.68</td>
<td>11.11%</td>
</tr>
<tr>
<td>950</td>
<td>20.71</td>
<td>2.15</td>
<td>9.08%</td>
</tr>
<tr>
<td>1000</td>
<td>24.02</td>
<td>2.57</td>
<td>9.37%</td>
</tr>
</tbody>
</table>
Table C.2: SPDY mode Non-SSL with Bandwidth fixed at 10 Mbps

<table>
<thead>
<tr>
<th>RTT [ms]</th>
<th>Mean PLT [sec]</th>
<th>St. dev</th>
<th>Rel. 95% CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>3.22</td>
<td>0.22</td>
<td>5.89%</td>
</tr>
<tr>
<td>100</td>
<td>3.93</td>
<td>0.29</td>
<td>6.50%</td>
</tr>
<tr>
<td>150</td>
<td>4.70</td>
<td>0.35</td>
<td>6.51%</td>
</tr>
<tr>
<td>200</td>
<td>5.65</td>
<td>0.52</td>
<td>8.14%</td>
</tr>
<tr>
<td>250</td>
<td>6.28</td>
<td>0.37</td>
<td>5.16%</td>
</tr>
<tr>
<td>300</td>
<td>6.95</td>
<td>0.76</td>
<td>9.57%</td>
</tr>
<tr>
<td>350</td>
<td>8.35</td>
<td>0.92</td>
<td>9.65%</td>
</tr>
<tr>
<td>400</td>
<td>8.54</td>
<td>0.70</td>
<td>7.14%</td>
</tr>
<tr>
<td>450</td>
<td>9.59</td>
<td>1.23</td>
<td>11.27%</td>
</tr>
<tr>
<td>500</td>
<td>9.59</td>
<td>1.23</td>
<td>11.27%</td>
</tr>
<tr>
<td>550</td>
<td>12.24</td>
<td>2.00</td>
<td>14.35%</td>
</tr>
<tr>
<td>600</td>
<td>12.24</td>
<td>1.31</td>
<td>9.37%</td>
</tr>
<tr>
<td>650</td>
<td>12.17</td>
<td>0.87</td>
<td>6.26%</td>
</tr>
<tr>
<td>700</td>
<td>13.75</td>
<td>1.96</td>
<td>12.49%</td>
</tr>
<tr>
<td>750</td>
<td>14.17</td>
<td>1.34</td>
<td>8.27%</td>
</tr>
<tr>
<td>800</td>
<td>15.07</td>
<td>0.86</td>
<td>5.01%</td>
</tr>
<tr>
<td>850</td>
<td>15.03</td>
<td>0.94</td>
<td>5.49%</td>
</tr>
<tr>
<td>900</td>
<td>18.12</td>
<td>1.61</td>
<td>7.81%</td>
</tr>
<tr>
<td>950</td>
<td>18.03</td>
<td>1.58</td>
<td>7.67%</td>
</tr>
<tr>
<td>1000</td>
<td>17.74</td>
<td>1.51</td>
<td>7.48%</td>
</tr>
</tbody>
</table>
Table C.3: HTTP mode SSL with Bandwidth fixed at 10 Mbps

<table>
<thead>
<tr>
<th>RTT[ms]</th>
<th>Mean PLT[sec]</th>
<th>St.dev</th>
<th>Rel. 95% CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>3.51</td>
<td>0.37</td>
<td>9.24%</td>
</tr>
<tr>
<td>100</td>
<td>4.34</td>
<td>0.54</td>
<td>10.90%</td>
</tr>
<tr>
<td>150</td>
<td>4.98</td>
<td>0.95</td>
<td>16.79%</td>
</tr>
<tr>
<td>200</td>
<td>5.11</td>
<td>0.51</td>
<td>8.82%</td>
</tr>
<tr>
<td>250</td>
<td>6.18</td>
<td>0.47</td>
<td>6.68%</td>
</tr>
<tr>
<td>300</td>
<td>6.65</td>
<td>0.35</td>
<td>4.68%</td>
</tr>
<tr>
<td>350</td>
<td>7.82</td>
<td>0.26</td>
<td>2.91%</td>
</tr>
<tr>
<td>400</td>
<td>8.18</td>
<td>0.39</td>
<td>4.22%</td>
</tr>
<tr>
<td>450</td>
<td>8.62</td>
<td>0.99</td>
<td>10.07%</td>
</tr>
<tr>
<td>500</td>
<td>9.80</td>
<td>0.98</td>
<td>8.73%</td>
</tr>
<tr>
<td>550</td>
<td>12.06</td>
<td>1.34</td>
<td>9.75%</td>
</tr>
<tr>
<td>600</td>
<td>12.02</td>
<td>1.52</td>
<td>11.11%</td>
</tr>
<tr>
<td>650</td>
<td>12.74</td>
<td>1.11</td>
<td>7.67%</td>
</tr>
<tr>
<td>700</td>
<td>12.24</td>
<td>1.00</td>
<td>7.13%</td>
</tr>
<tr>
<td>750</td>
<td>13.61</td>
<td>1.56</td>
<td>10.07%</td>
</tr>
<tr>
<td>800</td>
<td>14.02</td>
<td>0.74</td>
<td>4.66%</td>
</tr>
<tr>
<td>850</td>
<td>15.17</td>
<td>1.02</td>
<td>5.91%</td>
</tr>
<tr>
<td>900</td>
<td>17.71</td>
<td>2.53</td>
<td>12.50%</td>
</tr>
<tr>
<td>950</td>
<td>16.41</td>
<td>1.70</td>
<td>9.08%</td>
</tr>
<tr>
<td>1000</td>
<td>21.99</td>
<td>1.93</td>
<td>7.71%</td>
</tr>
</tbody>
</table>
Table C.4: HTTP mode Non-SSL with Bandwidth fixed at 10 Mbps

<table>
<thead>
<tr>
<th>RTT[ms]</th>
<th>Mean PLT[sec]</th>
<th>St.dev</th>
<th>Rel. 95% CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>2.95</td>
<td>0.19</td>
<td>5.68%</td>
</tr>
<tr>
<td>100</td>
<td>3.65</td>
<td>0.18</td>
<td>4.29%</td>
</tr>
<tr>
<td>150</td>
<td>4.55</td>
<td>0.31</td>
<td>5.98%</td>
</tr>
<tr>
<td>200</td>
<td>4.90</td>
<td>0.17</td>
<td>2.95%</td>
</tr>
<tr>
<td>250</td>
<td>5.29</td>
<td>0.55</td>
<td>9.09%</td>
</tr>
<tr>
<td>300</td>
<td>6.79</td>
<td>0.71</td>
<td>9.21%</td>
</tr>
<tr>
<td>350</td>
<td>7.40</td>
<td>0.95</td>
<td>11.24%</td>
</tr>
<tr>
<td>400</td>
<td>7.63</td>
<td>0.59</td>
<td>6.79%</td>
</tr>
<tr>
<td>450</td>
<td>8.39</td>
<td>1.05</td>
<td>11.01%</td>
</tr>
<tr>
<td>500</td>
<td>8.93</td>
<td>1.43</td>
<td>14.06%</td>
</tr>
<tr>
<td>550</td>
<td>9.87</td>
<td>0.28</td>
<td>2.45%</td>
</tr>
<tr>
<td>600</td>
<td>11.78</td>
<td>1.52</td>
<td>11.28%</td>
</tr>
<tr>
<td>650</td>
<td>11.72</td>
<td>1.90</td>
<td>14.18%</td>
</tr>
<tr>
<td>700</td>
<td>11.20</td>
<td>1.65</td>
<td>12.91%</td>
</tr>
<tr>
<td>750</td>
<td>11.37</td>
<td>2.31</td>
<td>17.78%</td>
</tr>
<tr>
<td>800</td>
<td>12.14</td>
<td>0.73</td>
<td>5.29%</td>
</tr>
<tr>
<td>850</td>
<td>13.79</td>
<td>1.52</td>
<td>9.68%</td>
</tr>
<tr>
<td>900</td>
<td>13.55</td>
<td>1.38</td>
<td>8.93%</td>
</tr>
<tr>
<td>950</td>
<td>15.15</td>
<td>1.81</td>
<td>10.47%</td>
</tr>
<tr>
<td>1000</td>
<td>16.41</td>
<td>1.77</td>
<td>9.46%</td>
</tr>
</tbody>
</table>
### Table C.5: SPDY mode SSL with RTT fixed at 150 ms

<table>
<thead>
<tr>
<th>Download BW[kbps]</th>
<th>Mean PLT[sec]</th>
<th>St.dev</th>
<th>Rel. 95% CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>25.67</td>
<td>25.67</td>
<td>3.11%</td>
</tr>
<tr>
<td>650</td>
<td>7.39</td>
<td>7.39</td>
<td>3.24%</td>
</tr>
<tr>
<td>1150</td>
<td>5.83</td>
<td>5.83</td>
<td>7.97%</td>
</tr>
<tr>
<td>1650</td>
<td>5.88</td>
<td>5.88</td>
<td>9.39%</td>
</tr>
<tr>
<td>2150</td>
<td>5.32</td>
<td>5.32</td>
<td>9.22%</td>
</tr>
<tr>
<td>2650</td>
<td>5.41</td>
<td>5.41</td>
<td>7.08%</td>
</tr>
<tr>
<td>3150</td>
<td>5.39</td>
<td>5.39</td>
<td>5.55%</td>
</tr>
<tr>
<td>3650</td>
<td>5.05</td>
<td>5.05</td>
<td>8.71%</td>
</tr>
<tr>
<td>4150</td>
<td>5.17</td>
<td>5.17</td>
<td>8.43%</td>
</tr>
<tr>
<td>4650</td>
<td>5.76</td>
<td>5.76</td>
<td>11.91%</td>
</tr>
<tr>
<td>5150</td>
<td>5.65</td>
<td>5.65</td>
<td>10.07%</td>
</tr>
<tr>
<td>5650</td>
<td>5.89</td>
<td>5.89</td>
<td>31.82%</td>
</tr>
<tr>
<td>6150</td>
<td>6.33</td>
<td>6.33</td>
<td>23.60%</td>
</tr>
<tr>
<td>6650</td>
<td>5.26</td>
<td>5.26</td>
<td>7.46%</td>
</tr>
<tr>
<td>7150</td>
<td>5.54</td>
<td>5.54</td>
<td>7.03%</td>
</tr>
<tr>
<td>7650</td>
<td>5.39</td>
<td>5.39</td>
<td>3.82%</td>
</tr>
<tr>
<td>8150</td>
<td>5.52</td>
<td>5.52</td>
<td>11.07%</td>
</tr>
<tr>
<td>8650</td>
<td>5.09</td>
<td>5.09</td>
<td>6.20%</td>
</tr>
<tr>
<td>9150</td>
<td>5.23</td>
<td>5.23</td>
<td>6.94%</td>
</tr>
<tr>
<td>9650</td>
<td>5.07</td>
<td>5.07</td>
<td>8.39%</td>
</tr>
<tr>
<td>10150</td>
<td>5.50</td>
<td>5.50</td>
<td>8.13%</td>
</tr>
<tr>
<td>10650</td>
<td>5.14</td>
<td>5.14</td>
<td>8.83%</td>
</tr>
<tr>
<td>11150</td>
<td>5.19</td>
<td>5.19</td>
<td>4.60%</td>
</tr>
<tr>
<td>11650</td>
<td>5.07</td>
<td>5.07</td>
<td>3.33%</td>
</tr>
<tr>
<td>12150</td>
<td>5.10</td>
<td>5.10</td>
<td>6.74%</td>
</tr>
<tr>
<td>12650</td>
<td>5.60</td>
<td>5.60</td>
<td>12.54%</td>
</tr>
<tr>
<td>13150</td>
<td>8.73</td>
<td>8.73</td>
<td>3.75%</td>
</tr>
<tr>
<td>13650</td>
<td>5.46</td>
<td>5.46</td>
<td>12.95%</td>
</tr>
<tr>
<td>14150</td>
<td>5.46</td>
<td>5.46</td>
<td>9.67%</td>
</tr>
<tr>
<td>14650</td>
<td>5.46</td>
<td>5.46</td>
<td>9.80%</td>
</tr>
<tr>
<td>15150</td>
<td>5.00</td>
<td>5.00</td>
<td>2.95%</td>
</tr>
</tbody>
</table>
Table C.6: SPDY mode Non-SSL with RTT fixed at 150 ms

<table>
<thead>
<tr>
<th>Download BW[kbps]</th>
<th>Mean PLT[sec]</th>
<th>St.dev</th>
<th>Rel. 95% CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>23.20</td>
<td>2.46</td>
<td>9.28%</td>
</tr>
<tr>
<td>650</td>
<td>6.41</td>
<td>0.34</td>
<td>4.61%</td>
</tr>
<tr>
<td>1150</td>
<td>5.47</td>
<td>0.20</td>
<td>3.13%</td>
</tr>
<tr>
<td>1650</td>
<td>5.01</td>
<td>0.29</td>
<td>5.05%</td>
</tr>
<tr>
<td>2150</td>
<td>4.92</td>
<td>0.23</td>
<td>4.15%</td>
</tr>
<tr>
<td>2650</td>
<td>4.87</td>
<td>0.53</td>
<td>9.49%</td>
</tr>
<tr>
<td>3150</td>
<td>4.89</td>
<td>0.20</td>
<td>3.64%</td>
</tr>
<tr>
<td>3650</td>
<td>4.51</td>
<td>0.26</td>
<td>5.06%</td>
</tr>
<tr>
<td>4150</td>
<td>4.69</td>
<td>0.10</td>
<td>1.96%</td>
</tr>
<tr>
<td>4650</td>
<td>4.71</td>
<td>0.41</td>
<td>7.56%</td>
</tr>
<tr>
<td>5150</td>
<td>4.72</td>
<td>0.30</td>
<td>5.52%</td>
</tr>
<tr>
<td>5650</td>
<td>4.96</td>
<td>0.63</td>
<td>11.06%</td>
</tr>
<tr>
<td>6150</td>
<td>4.78</td>
<td>0.50</td>
<td>9.13%</td>
</tr>
<tr>
<td>6650</td>
<td>4.55</td>
<td>0.11</td>
<td>2.13%</td>
</tr>
<tr>
<td>7150</td>
<td>4.55</td>
<td>0.29</td>
<td>5.56%</td>
</tr>
<tr>
<td>7650</td>
<td>4.71</td>
<td>0.25</td>
<td>4.66%</td>
</tr>
<tr>
<td>8150</td>
<td>4.80</td>
<td>0.22</td>
<td>4.07%</td>
</tr>
<tr>
<td>8650</td>
<td>4.61</td>
<td>0.29</td>
<td>5.57%</td>
</tr>
<tr>
<td>9150</td>
<td>4.69</td>
<td>0.25</td>
<td>4.71%</td>
</tr>
<tr>
<td>9650</td>
<td>4.60</td>
<td>0.21</td>
<td>3.95%</td>
</tr>
<tr>
<td>10150</td>
<td>4.70</td>
<td>0.35</td>
<td>6.51%</td>
</tr>
<tr>
<td>10650</td>
<td>4.73</td>
<td>0.38</td>
<td>7.07%</td>
</tr>
<tr>
<td>11150</td>
<td>4.55</td>
<td>0.22</td>
<td>4.22%</td>
</tr>
<tr>
<td>11650</td>
<td>4.68</td>
<td>0.28</td>
<td>5.16%</td>
</tr>
<tr>
<td>12150</td>
<td>4.80</td>
<td>0.32</td>
<td>5.85%</td>
</tr>
<tr>
<td>12650</td>
<td>4.68</td>
<td>0.21</td>
<td>3.89%</td>
</tr>
<tr>
<td>13150</td>
<td>5.04</td>
<td>0.36</td>
<td>6.27%</td>
</tr>
<tr>
<td>13650</td>
<td>4.59</td>
<td>0.12</td>
<td>2.25%</td>
</tr>
<tr>
<td>14150</td>
<td>4.70</td>
<td>0.30</td>
<td>5.52%</td>
</tr>
<tr>
<td>14650</td>
<td>4.73</td>
<td>0.42</td>
<td>7.87%</td>
</tr>
<tr>
<td>15150</td>
<td>4.54</td>
<td>0.12</td>
<td>2.41%</td>
</tr>
</tbody>
</table>
Table C.7: HTTP mode SSL with RTT fixed at 150 ms

<table>
<thead>
<tr>
<th>Download BW[kbps]</th>
<th>Mean PLT[sec]</th>
<th>St.dev</th>
<th>Rel. 95% CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>27.02</td>
<td>1.54</td>
<td>4.98%</td>
</tr>
<tr>
<td>650</td>
<td>11.18</td>
<td>2.80</td>
<td>21.92%</td>
</tr>
<tr>
<td>1150</td>
<td>6.86</td>
<td>0.87</td>
<td>11.06%</td>
</tr>
<tr>
<td>1650</td>
<td>5.27</td>
<td>0.21</td>
<td>3.42%</td>
</tr>
<tr>
<td>2150</td>
<td>5.70</td>
<td>0.74</td>
<td>11.42%</td>
</tr>
<tr>
<td>2650</td>
<td>5.27</td>
<td>0.63</td>
<td>10.43%</td>
</tr>
<tr>
<td>3150</td>
<td>5.27</td>
<td>1.37</td>
<td>22.84%</td>
</tr>
<tr>
<td>3650</td>
<td>4.81</td>
<td>0.49</td>
<td>8.84%</td>
</tr>
<tr>
<td>4150</td>
<td>5.09</td>
<td>0.59</td>
<td>10.10%</td>
</tr>
<tr>
<td>4650</td>
<td>4.68</td>
<td>0.26</td>
<td>4.79%</td>
</tr>
<tr>
<td>5150</td>
<td>4.68</td>
<td>0.47</td>
<td>8.73%</td>
</tr>
<tr>
<td>5650</td>
<td>4.59</td>
<td>0.35</td>
<td>6.62%</td>
</tr>
<tr>
<td>6150</td>
<td>4.21</td>
<td>0.21</td>
<td>4.39%</td>
</tr>
<tr>
<td>6650</td>
<td>4.84</td>
<td>0.93</td>
<td>16.81%</td>
</tr>
<tr>
<td>7150</td>
<td>4.71</td>
<td>0.81</td>
<td>15.08%</td>
</tr>
<tr>
<td>7650</td>
<td>4.46</td>
<td>0.14</td>
<td>2.78%</td>
</tr>
<tr>
<td>8150</td>
<td>5.60</td>
<td>0.93</td>
<td>14.51%</td>
</tr>
<tr>
<td>8650</td>
<td>4.49</td>
<td>0.08</td>
<td>1.50%</td>
</tr>
<tr>
<td>9150</td>
<td>4.15</td>
<td>0.19</td>
<td>4.10%</td>
</tr>
<tr>
<td>9650</td>
<td>4.49</td>
<td>0.27</td>
<td>5.24%</td>
</tr>
<tr>
<td>10150</td>
<td>4.54</td>
<td>0.36</td>
<td>6.96%</td>
</tr>
<tr>
<td>10650</td>
<td>5.39</td>
<td>0.81</td>
<td>13.24%</td>
</tr>
<tr>
<td>11150</td>
<td>4.46</td>
<td>0.31</td>
<td>5.99%</td>
</tr>
<tr>
<td>11650</td>
<td>4.32</td>
<td>0.35</td>
<td>7.19%</td>
</tr>
<tr>
<td>12150</td>
<td>4.47</td>
<td>0.13</td>
<td>2.63%</td>
</tr>
<tr>
<td>12650</td>
<td>4.49</td>
<td>0.40</td>
<td>7.90%</td>
</tr>
<tr>
<td>13150</td>
<td>5.11</td>
<td>0.46</td>
<td>7.94%</td>
</tr>
<tr>
<td>13650</td>
<td>4.50</td>
<td>0.21</td>
<td>4.03%</td>
</tr>
<tr>
<td>14150</td>
<td>4.59</td>
<td>0.31</td>
<td>5.97%</td>
</tr>
<tr>
<td>14650</td>
<td>4.60</td>
<td>0.48</td>
<td>9.17%</td>
</tr>
<tr>
<td>15150</td>
<td>4.51</td>
<td>0.48</td>
<td>9.31%</td>
</tr>
</tbody>
</table>
Table C.8: HTTP mode Non-SSL with RTT fixed at 150 ms

<table>
<thead>
<tr>
<th>Download BW [kbps]</th>
<th>Mean PLT [sec]</th>
<th>St. dev</th>
<th>Rel. 95% CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>27.02</td>
<td>1.54</td>
<td>4.98%</td>
</tr>
<tr>
<td>650</td>
<td>11.18</td>
<td>2.80</td>
<td>21.92%</td>
</tr>
<tr>
<td>1150</td>
<td>6.86</td>
<td>0.87</td>
<td>11.06%</td>
</tr>
<tr>
<td>1650</td>
<td>5.27</td>
<td>0.21</td>
<td>3.42%</td>
</tr>
<tr>
<td>2150</td>
<td>5.70</td>
<td>0.74</td>
<td>11.42%</td>
</tr>
<tr>
<td>2650</td>
<td>5.27</td>
<td>0.63</td>
<td>10.43%</td>
</tr>
<tr>
<td>3150</td>
<td>5.27</td>
<td>1.37</td>
<td>22.84%</td>
</tr>
<tr>
<td>3650</td>
<td>4.81</td>
<td>0.49</td>
<td>8.84%</td>
</tr>
<tr>
<td>4150</td>
<td>5.09</td>
<td>0.59</td>
<td>10.10%</td>
</tr>
<tr>
<td>4650</td>
<td>4.68</td>
<td>0.26</td>
<td>4.79%</td>
</tr>
<tr>
<td>5150</td>
<td>4.68</td>
<td>0.47</td>
<td>8.73%</td>
</tr>
<tr>
<td>5650</td>
<td>4.59</td>
<td>0.35</td>
<td>6.62%</td>
</tr>
<tr>
<td>6150</td>
<td>4.21</td>
<td>0.21</td>
<td>4.39%</td>
</tr>
<tr>
<td>6650</td>
<td>4.84</td>
<td>0.93</td>
<td>16.81%</td>
</tr>
<tr>
<td>7150</td>
<td>4.71</td>
<td>0.81</td>
<td>15.08%</td>
</tr>
<tr>
<td>7650</td>
<td>4.46</td>
<td>0.14</td>
<td>2.78%</td>
</tr>
<tr>
<td>8150</td>
<td>5.60</td>
<td>0.93</td>
<td>14.51%</td>
</tr>
<tr>
<td>8650</td>
<td>4.49</td>
<td>0.08</td>
<td>1.50%</td>
</tr>
<tr>
<td>9150</td>
<td>4.15</td>
<td>0.19</td>
<td>4.10%</td>
</tr>
<tr>
<td>9650</td>
<td>4.49</td>
<td>0.27</td>
<td>5.24%</td>
</tr>
<tr>
<td>10150</td>
<td>4.54</td>
<td>0.36</td>
<td>6.96%</td>
</tr>
<tr>
<td>10650</td>
<td>5.39</td>
<td>0.81</td>
<td>13.24%</td>
</tr>
<tr>
<td>11150</td>
<td>4.46</td>
<td>0.31</td>
<td>5.99%</td>
</tr>
<tr>
<td>11650</td>
<td>4.32</td>
<td>0.35</td>
<td>7.19%</td>
</tr>
<tr>
<td>12150</td>
<td>4.47</td>
<td>0.13</td>
<td>2.63%</td>
</tr>
<tr>
<td>12650</td>
<td>4.49</td>
<td>0.40</td>
<td>7.90%</td>
</tr>
<tr>
<td>13150</td>
<td>5.11</td>
<td>0.46</td>
<td>7.94%</td>
</tr>
<tr>
<td>13650</td>
<td>4.50</td>
<td>0.21</td>
<td>4.03%</td>
</tr>
<tr>
<td>14150</td>
<td>4.59</td>
<td>0.31</td>
<td>5.97%</td>
</tr>
<tr>
<td>14650</td>
<td>4.60</td>
<td>0.48</td>
<td>9.17%</td>
</tr>
<tr>
<td>15150</td>
<td>4.51</td>
<td>0.48</td>
<td>9.31%</td>
</tr>
</tbody>
</table>
Bibliography


Bibliography


