A Latency-Determining/User Directed Firefox Browser Extension

Philip Avery Mein

Submitted to the graduate degree program Electrical Engineering & Computer Science and The Graduate Faculty of the University of Kansas School of Engineering in partial fulfillment of the requirements for the degree of Master of Science

Thesis Committee:

Dr. James P.G. Sterbenz: Chairperson

Dr. Bo Luo

Dr. Gary J. Minden

Date Defended

The Thesis Committee for Philip Avery Mein certifies
that this is the approved version of the following thesis:

A Latency-Determining/User Directed Firefox Browser Extension

Committee:

Chairperson

Date Approved

Abstract

As the World Wide Web continues to evolve as the preferred choice for information access it is critical that its utility to the user remains. Latency as a result of network congestion, bandwidth availability, server processing delays, embedded objects, and transmission delays and errors can impact the utility of the web browser application. To improve the overall user experience the application needs to not only provide feedback to the end user about the latency of links that are available but to also provide them controls in the retrieval of the web content. This thesis presents a background and related work relating to latency and web optimization techniques to reduce this latency and then introduce an improvement to the "latency aware" Mozilla Firefox extension which was originally developed by Sterbenz et. al., in 2002. This these describes the architecture and prototype implementation, followed with an analysis of its effectiveness to predict latency and future work.

Key Terms- high speed, mobile, wireless, weakly-connected, information access, web browsing, caching, firefox-addon, latency

For my family, whose love and support throughout the course of this thesis meant sacrifice on their part.

Acknowledgements

First and foremost I offer my sincerest gratitude to my supervisor, Dr. James Sterbenz, who has provided support to my efforts throughout my thesis with his knowledge and experience. During my course work and initial thesis research he was flexibile while my family and I were going through two different moves and job changes. As a member of the University of Kansas faculty Dr. Sterbenz shared his learned insights to high-performance networks that provided me a new perspective for my own experience.

Dr. Hossein Saiedian, of the Department of Electrical Engineering & Computer Science for his guideance as I entered the Masters program.

The ResiliNets gradutate namely Egemen Cetinkaya for his assistance on the tools used to compile my results and being there to provide honest feedback.

My deepest thanks to my family for their unwaivering love and support throught my life.

Last but certainly not least my wife, Christy. She was always there cheering me up and when I needed it pushing me to the finish line.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

The modern Internet has seen marked improvements in its responsiveness but even minor variations in this responsiveness may impact productivity and as a result incur associated costs. Adding even frations of a second to response time has shown to have an adverse effect on human behavior [1]. Latency and its affect on users has been studied in [2], [3], [4], [5], [6], and [7]. When latency is low and there is a seamless transition from when a user requests a web page and the response is displayed increases productivity, reduces costs, and maintains user satisfaction [2]. IBM experimented with this theory and demonstrated when a system's response time was reduced from three seconds to sub-second, response time increased a programmer's productivity by 106% [2]. Another study by Zona Research showed that a decrease in the load time of a web page by one second decreased the bailout behavior of users visiting the page from 30% to 6% [3]. It is true that time on the Web does equate to real money. Cloud providers are taking this to extremes cutting out unneeded content in web pages to conserve valuable milliseconds [7]. Cloud providers are employing strategies to reduce page file size, for example stripping JavaScript of comments, using shorter variable names, preoptimizing images, and using JavaScript Object Notation (JSON) rather than the wordy eXtensible Markup Language (XML) [8].

Over the years, the Web has evolved and its workload today can be characterized more precisely as an interactive application. To maintain the usefulness of this application it must provide response times to the user of 100 ms or less [9]. Unfortunately the web browsing application is prone to many different types of delay that have a cumulative affect to impose latency on the system. For simplicity, these contributing sources of web browsing latency include the web server, the web browser, and the network that sits between them. Increases in bandwidth and improvements to the network infrastructure has driven down the overall latency for a web page. According to Verizon, their broadband network, in 2010, averaged 42 ms of latency for US intra-continental traffic [10]. This represents the latency within their broadband network to propogate a packet of data from the source to its destination. This does not account for any processing delays at the source or destination. There could also be multiple round trip time (RTT) delays when rendering a web page with numerous embedded objects. Although deployment of broadband access has increased throughout the US, latency can still be experienced within the application. When one considers the push towards wireless and cellular network access that is now becoming more the norm, the links that individuals are using are suseptable to being weak, intermittent, and error prone. This means that latency will continue to be a factor in the application's usefulness.

1.1 Contribution

The contribution of this thesis is to improve the utility of the web browser application. The WVM extension developed for this thesis provides users information about the latecny of links and provides a mechanism to take user directed action when navigating links that are highly latent. Current browsers do not provide information to the user about the latency of links on a web page nor do they give the user the ability to take appropriate actions when latency is detected. In order to preserve the usefulness of the application it is desirable to provide a feedback and control mechanism within the web browser to proactively notify the user of delays and also provide them with alternate behaviors in dealing with the delays. To address this gap, a framework was built upon a Firefox extension that not only provides the user with latency feedback but also offers tools to take user directed actions to deal with the latency. The Firefox WVM extension described herein expands and optimizes the original WVM prototype application developed by Sterbenz et. al [9]. The remainder of this thesis is organized as follows: Chapter 2 reviews the background and related work. Chapter 3 describes the design and implementation, which is the WVM Firefox extension. Chapter 4 provides the methodology and analysis of our experimental results and finally, Chapter 5 concludes the paper wtih contributions and future work.

Chapter 2

Background and Related Work

This chapter provides an overview of concepts relavent to this research including their background, related work, and other items of interest. Section 2.1 explains the different contributors to delay in the web browsing environment. Section 2.2 reviews techniques and related work that aims to minimize or elliminate the preceived latency within the web browsing application. Section 2.3 discusses the improvements and features of modern web browsers to address latency and performance. Finally, Section 2.4 describes the landscape of broadband access, current benchmarked latencies, and other measured and reported metrics that influence the responsiveness of web browsing.

2.1 Latency in Web Browsing

The workload of modern web browsing can be described as an interactive application, in which users interact with text, images, video, and other information in a hyperlinked fashion. These hyperlinks may link to additional content on the same server or may redirect the client to a far off location. A potential problem with web browsing as an interactive application is its susceptibility to latency that affects its responsiveness. For the sake of clarity, in this thesis the response time is defined as the time that elapses from when a web page is requested until the first byte is received. The contributors to latency can be boiled down to, the client (web browser), the network, the protocol used, and the server. Thinking in these terms makes it easier to define the latencies and also to discuss the methods that have been researched and employed to address them. This research is important because web browsers do not provide information to the user about the latency associated with the link they are following. Response time is the most straightforward predictor of user satisfaction [1] and without providing feedback to user and the tools to avoid it will diminish the utility of the web browser application. In today's environment of abundant wireless and mobile devices, accessing web applications in challenged areas with limited bandwidth, episodic connections, error prone channels, the need for this information is only exaggerated.

Working from the client to the server a typical web browsing application will access a hyperlink which will in turn cause the client to make a request across an internet of links and nodes. These links and nodes will be providing service at varying speeds (bandwidth) and operating under differing degrees of congestion and errors. Upon arrival at the remote server, this request may undergo additional processing due to its static or dynamic nature. When the request is completed at the server, the response must traverse the internet of links and nodes once again back to the client which may need to make subsequent requests for embedded objects. This environment produces delay in the form of processing delay (both client and server), propagation delay, queueing delays, transmission delay, and delays from retransmissions and lossy connectivity.

2.1.1 Processing delay

Processing delay can occur at either the client or the server. At the client the processing delay that adds to the overall latency is due to the rendering engine built into the web browser. The primary job of the rendering engine is to display the web page once it is received. The study performed by Nielson et. al, measured the responsiveness of the leading rendering engines, Gecko, Trident, Presto, and WebCore [11]. Improvements to reduce delays in these engines are driven by the developers and their need to deliver a more desirable product. The overhead involved in displaying a web page at the client was sourced to client side scripting (i.e. JavaScript) and the rendering or layout itself [11]. The processing delay on the server is the time from when the server receives the request to the time the server starts to transmit the reply. Factors that can influence the severity of this delay include the current load of the server responding to multiple requests, the preprocessing of a dynamic web page (i.e. PHP, ASP, and JSP), processing of complex embedded objects, encryption, and the need to query data from external sources. Moving from static web content to dynamic web pages and web applications will have a net increase to the processing delay and overall latency. Technologies and research to combat server processing delay include load balancing, prefetching, caching, SSL certificate off loading, and content delivery networks (CDNs).

2.1.2 Propagation delay

The propagation characteristics of the network are sometimes confused with or mixed in with the concept of bandwidth. A bandwidth is the capacity or bit rate of the communication channel. This is typically described in bits / second or mltiples of it (Mb/s or Gb/s). In networking, propagation delay is the time it takes to transmit data a set distance. A simple formula can be used to compute the propagation delay for a given network to be d/s where d is the distance and s is the speed. The medium that the bytes are being transmitted across dictates an upper bound in respect to the speed at which the bytes are traveling. For modern high speed networking, this typically approaches the speed of light. Putting some real numbers to propagation delay and the Internet backbone, consider the time it takes to transmit data from San Francisco to New York (4125 km). The current latency reported by AT&T's broadband network is approximately 75 ms [12]. This delay includes the time for the data to travel the distance and traverse the equipment within the network. The networking equipment will add switching, queuing, and buffering delay. Now consider the impact of this latency in our global network using geosynchronous satellites or transatlantic fiber. The latency from San Francisco to Hong Kong is 160 ms [12]. For a person living in the Midwest, the total application latency associated to propagation delay may be minimal (35 ms [12]) due to the rather central geographic location and intra-continental traffic patterns. Residents of Hawaii do not have this same benefit and can expect a latency of 60 ms just to reach the first hop on the mainland [12]. Even if benefited from being centrally located to network servers, there is never a guarantee that data will follow in a reasonably straight path to its destination. Networks are at times policy routed Internet or tunneled through a Virtual Private

Network (VPN) that may route traffic to a home office before sending it back out to its actual destination. Propagation delay can be a larger multiplier in modern web applications that use multiple embedded objects and the potential round trip setup cost to transfer each object. Some of the same research areas that address processing delay have also been applied to propagation delay, caching, prefetching, and CDNs.

2.1.3 Transmission delay

Transmission delay is the time it takes to put the data onto the network. Depending on the type of medium being used, infrared, fiber optic, wireless, or copper, the speed can range from a few megabits per second (Mb/s) to multi-gigabits per second (Gb/s). This speed is described as bandwidth. Even in today's world with broadband residential connections and multi-gigabit network backbones, transmission delay can still be a source of latency for web applications. The Internet and the applications that run on top of it have evolved with the increased bandwidth. Applications are using more and more rich content. Higher resolution images, streaming video, on demand movies, and web conferencing have all become common. Netflix usage on the Internet is now at 29.7% of the consumed bandwidth during peak download times [13]. That is not the total video streaming usage of the Internet which is at 49.2% of the consumed bandwitdth during peak download time, but only Netflix [13]. This demonstrates that the bandwidth of the Internet can continue to increase but congestion and transmission delay will still need to be considered in designing network architecture. New technologies will be adopted to fill the newly provided resources. Data paths from the user to the destination travel over these shared links with the bandwidth being aggregate and competing alongside resource greedy applications. The bottom line is transmission delay is variable and can change depending on the end users current connection method (wireless, copper, infrared), the site they are attempting to access, the current usage by other applications along the path, and the congestion of routers in between.

2.1.4 Retransmission/Lossy Connectivity

The error rates and retransmissions that occur in networks are changing. For most, gone are the days of dialup analog connections that were prone to errors and lengthy retransmissions. The trouble in today's network environment is the adaption of mobile and wireless networking. Wireless and mobile networking are notorious for having episodic connections when they are in densely connected environments or are truly mobile. A user accessing the network with a PDA, cell phone or laptop that is in motion can move in and out of connectivity hot spots, compete in highly saturated environments for shared medium, or lose signal strength. This type of connectivity can cause delay while the network interface card negotiates with a new access point, waits for the shared resource, or requests lost or errored data packets that must be retransmitted. Wireless users that are at rest can still have connectivity issues due to the environment and suffer from similar retransmission issues causing delay in their web browsing application. As will be discussed later, [14] and [15] performed research in this space to measure and provide possible solutions to wireless network packet loss and retransmissions.

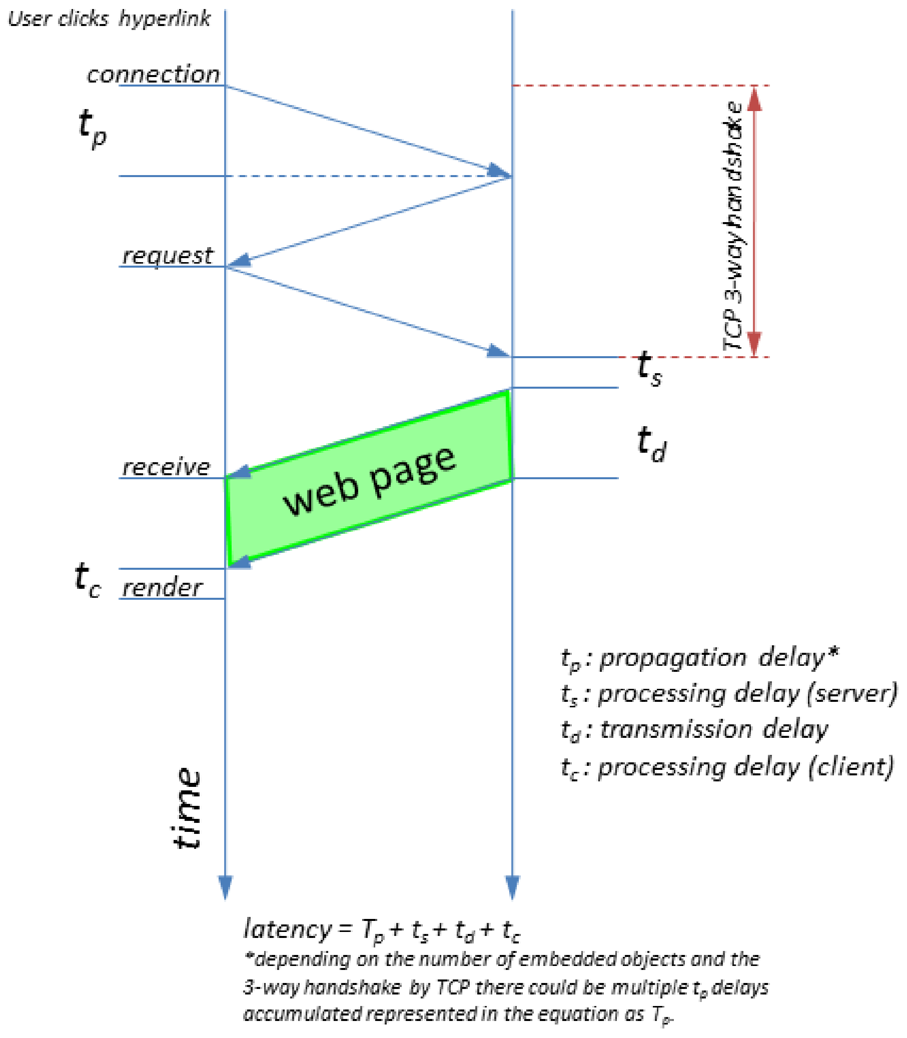

User clicks hyperlink
connection
t_p
request
TCP 3-way handshake
t_s
t_d
receive
web page
t_c
render
t_p: propagation delay*
t_s: processing delay (server)
t_d: transmission delay
t_c: processing delay (client)
time
latency = $T_p + t_s + t_d + t_c$
*depending on the number of embedded objects and the 3-way handshake by TCP there could be multiple t_p delays accumulated represented in the equation as T_p.

Figure 2.1. Flow diagram of web page request

2.2 Methods for Reducing Delay

The overall latency caused by these delays is cumulative and affects the overall utility of the web browsing application. Sterbenz [16] uses flow diagrams to illustrate these delays and their relationship to one another. Figure 2.1 shows the flow diagram for a standard web page request after a user clicks a hyperlink. The overall latency can be notated as: $latency = T_{\mathrm{p}} + t_{\mathrm{s}} + t_{\mathrm{d}} + t_{\mathrm{c}}$, where T_{p} represents the total propagation delay. The total propagation delay will depend on the number of round trips required to establish a connection and the queuing, switching and buffering delay in the network. The t_{s} represents the server side processing delay, t_{d} represents transmission delay, and t_{c} represents the client side processing delay. For each of the previously discussed latency contributors related work is presented with their solutions. Some of the research topics can be applied to suppress or reduce latency from multiple sources. Table 2.1 summarizes the research areas and which delays they may address. This Section will cover improvements to the client and server to avoid latency as a result of processing delay. The next Section will review the transmission delay associated with today's networks and their evolution. The final Section will focus on propagation delay and the popular techniques to counteract them to include caching and prefetching.

Table 2.1. Web Application latency Research

Research Area	$t_{\mathrm{c}}/t_{\mathrm{s}}$	t_{p}	t_{d}
Caching, CDNs	×	×	
Prefetching	×	×	×
Load balancing, Decomposition, SSL Offloading	×		
HTTP/TCP Modification		×	
Compression			×

2.2.1 Minimizing Processing Delay

Processing delay introduced at the client workstation and browser is an area where the research community can do little compared to the benefits of increased CPU power and improvements to the rendering engine. Increasing the CPU power of the client workstation will allow the page to be rendered more quickly and process the client side scripts faster. Improvements to the rendering engines by their respective developers have made them more efficient and have reduced their processing delay. Section 2.3 discusses the modern web browsers and their improvements to reduce processing delay. Processing delay on the server side has been a topic of research and solutions have been proposed. Load balancing, clustering, SSL offloading as well as caching and prefetching all help to reduce processing delay at the server. Load balancing and clustering provide additional processing power to service multiple user requests across multiple systems. An entire market has developed providing these types of solutions in the form of frontend appliances and the web server themselves. In order to speed response time specific services have also been isolated and removed from the web server. An example of a process that has been moved off the web server and onto its own platform is the encryption process that can be a burden on the web server. SSL offloading puts this computationally intensive operation onto a system optimized to perform this task [17].

2.2.2 Improving Bandwidth

The obvious answer to transmission delay and the bandwidth issue is to over provision whenever possible. This may solve the problem of bandwidth at the client side with dedicated resources but there still may be bandwidth issues in the network that they cannot control. Wireless networks pose another bandwidth issue where the resources are shared. Another solution to transmission delay is to send less data. Compression has been used to make more efficient use of available resources. [18] Disabling embedded objects to reduce the amount of content loaded with each page request can also be done. [19] found that as the number of embedded objects grew beyond the amount of configured parallelism in the browser the wait time grew due to queuing for retrieval. Offering web pages at different levels of quality to reduce bandwidth could also be done to reduce the amount of data transferred. An example could be to send images at lower resolutions, 320×240 instead of 640×480, providing alternate frame rates, 15 frames/s instead of 30 frames/s, and lowering the quality of the audio. As it was mentioned earlier, Netflix has grown to be the gorilla in the room being the major consumer of Internet resources. Only a few years ago it was peer to peer networking such as Gnutella that was the major consumer. What this tells us is that despite our improvements to infrastructure new technologies will take advantage of the resources and again drive transmission delay up and increase the latency associated to the application.

2.2.3 Caching and CDN Systems

In previous discussion it was mentioned that propagation delay is bound to physical laws that impose an upper bound to the speed at which the data can travel through the medium. Unable to circumvent this physical restriction in data transmissions the only option is to shorten the distance the data must travel. In web applications this can be done through the use of either a caching systems or CDN (Content Delivery Network). The types of web caches that are suited for this include local cache, proxy servers, hierarchical caches and a content delivery networks (CDNs). All are very similar in that they store a copy of the requested data closer to the client so that the distance and latency associated to it is reduced. Figure 2.2 demonstrates the location of the information in the different cache schemes relative to the web browser application. In the figure the initial requests are depicted in red and the subsequent requests in green. Starting at the top of the figure the local cache positions the content closest to the client. An initial request is made to the origin web server and the content is returned and stored in the web browser's local cache. The next request made for this content will be serviced by the local browser cache. By doing this the latency incurred in traversing the network is eliminated. The down side is that this cache is not shared with other web browsers. A proxy cache is a system implemented at the border of a network or within the network to service requests from multiple clients As requests are made the proxy server caches responses and may optionally return a cached version of the content directly and avoids sending the request to the origin server. Proxy servers have the advantage of storing results from requests made by multiple clients. This is particularly useful when multiple clients are using a shared proxy server where the first client may request a web page that is retrieved for the specified server but other clients that make the same request are now given the cached version. A hierarchical cache is a special type of proxy cache that tiers the caching. These cache tiers work similar to DNS (Domain Name Service). The requests are referred through the network until either the content is found or the origin server is reached. As the content is returned each cache in the path makes a copy of the content for potential future requests. Che et. al, researched the design and performance results of this type of cache system [20]. For a cache system to be efficient both the size of the client community and the hit ratio are increased. Finally, CDNs distribute and bring the data closer to the web browser like a hierarchical web cache system. The difference between a CDN and a hierarchical cache system is that CDNs pre-position content from an origin source to multiple geographically dispersed locations on the network. A client may use any of the CDN servers to retrieve content through the mechanics of content routing. In summary CDNs are typically a commercial service provided

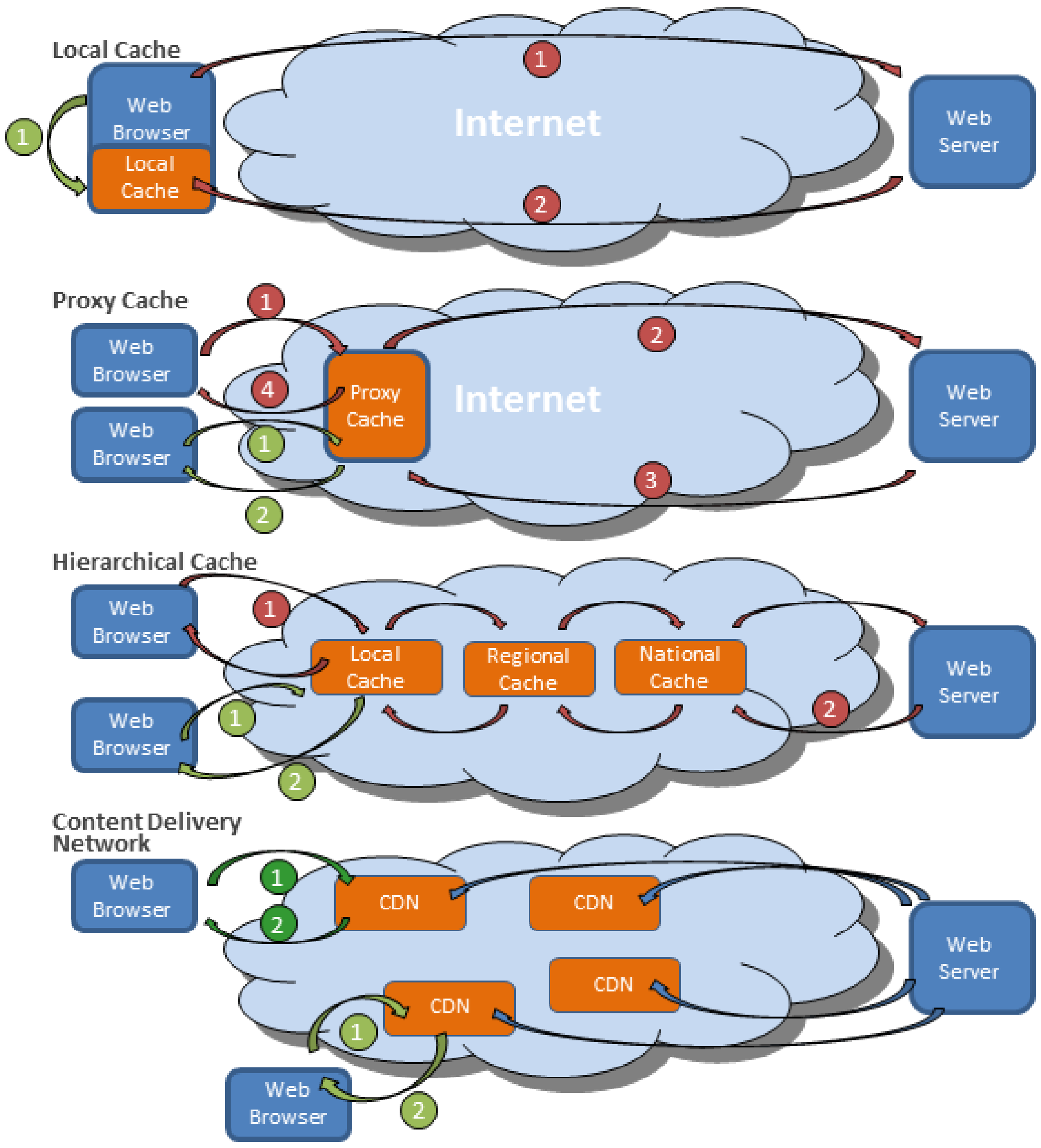


Figure 2.2. Types of caching systems

to enhance the performance of web sites. A local cache operates much in the same way as a proxy server but without sharing the cached content with multiple clients. The local cache is typically managed by the web browsing application itself and uses local resources to store copies of the content. Hierarchical caches are tiered proxy caches. The drawback to any cache is that it may become stale. When this happens content has changed since it was cached and the user may use the cached version without getting the more recent copy. In all cases of using cached data the web browser typically provides no indication that a cached version is being displayed in place of the source data. This is a growing issue as web pages are more dynamic.

2.2.4 Prefetching

In order to reduce the effects of intermittent connectivity and also reduce the perceived latency of remote systems prefetching can be implemented. The concept of prefetching and its benefit is that after retrieving a web page a user will spend time viewing the page before either following a link or navigating to another page [21]. During this idle time of viewing the page the prefetch engine could use the idle system to retrieve additional web pages. Also in an environment where the web browser becomes disconnected the user may request content that was prefetched while they were connected and thus be able to display it without latency. The idea would be to prefetch as much content while connected (horde) so as to increase the likelihood that content the user requests next is available.

Prefetching algorithms identify links on the current web page or infer links based on the page being viewed. Depending on their likelihood of being followed, this is known as the access probability; the content for these links is then retrieved before the link is actually requested by the user. [22] studied the effectiveness of client-side prefetching and found that latency could be reduced by as much as 81%. It is important to point out that the most significant part of a prefetching engine is its predictor that determines which links will be prefetched. With poor prediction prefetching would consume network resources requesting content that may not be needed by the user. The research by [5] evaluated some of the well-known prefetching algorithms not only from a hit ratio perspective but from a user perceived latency. The research evaluated algorithms that used Prediction by Partial Match (PPM) [23–25] and algorithms that used Dependency Graphs [26, 27]. In less aggressive prefetching conditions the Dependency Graph algorithms outperformed the PPM algorithms.

2.2.5 TCP Improvements

There have been many studies done on the TCP protocol and the characteristics which impact the overall latency of web browsing. [28], [29], [30], and [31] have all explained the negative effects of the congestion window size in the early stages of the TCP slow-start phase. The TCP slow-start model is designed to perform optimally for long sustained connections to a single host. As [31] points out TCP is not suited for the frequent, short, request-response traffic of HTTP. This behavior of HTTP impacts the performance and perceived latency of the web browser because connections that have paid the latency cost of the 3-way handshake and have started to grow the congestion window are now discarded as new requests are made to a new host. To correct this problem [28] suggested increasing the TCP initial congestion window size from three segments to at least ten. This change could improve the average latency of HTTP responses by 10%.

Another aspect of the TCP protocol that has warranted study is the 3-way handshake. [32] proposes a way to reduce the latency incurred by the 3-way handshake known as TCP splitting. Much like a proxy server or CDN where content is placed closer to the client, TCP splitting uses a proxy server positioned closer to the client for TCP connection establishment and segment acknowledgement. The HTTP GET using this proxy server configuration between the client and the web server can be seen in Figure 2.3. The RTT to establish the 3-way handshake is shorten from y ms to x ms. In addition, if the proxy server has an established TCP connection to the web server that has finished its slow start phase it can possibly retrieve the request from the web server in a single segment. With the response cached at the proxy server acknowledgement latency is shortened and the client grows its congestion window size at a much faster rate.

2.3 Modern Web Browsers

A few of the most notable web browsers available today are Mozilla Firefox [33], Microsoft Internet Explorer [34], Google Chrome [35], and Safari [36]. These web browsers compete to provide a more desirable user experience. A major factor in the user satisfaction is the rendering time from click to presentation of the requested web page. The underlying rendering engines mentioned in Section 2.1.1 are responsible for much of this function. Supported features in modern browsers to aid the rendering time can include: local cache, prefetching, configurable proxy servers, compression, HTTP and network protocol enhancements, and improved JavaScript interpreters. Although each of the browsers and their respected rendering engines support many of these features it is important to point out some specific features, default settings and behaviors. Powell [37] does an excellent job

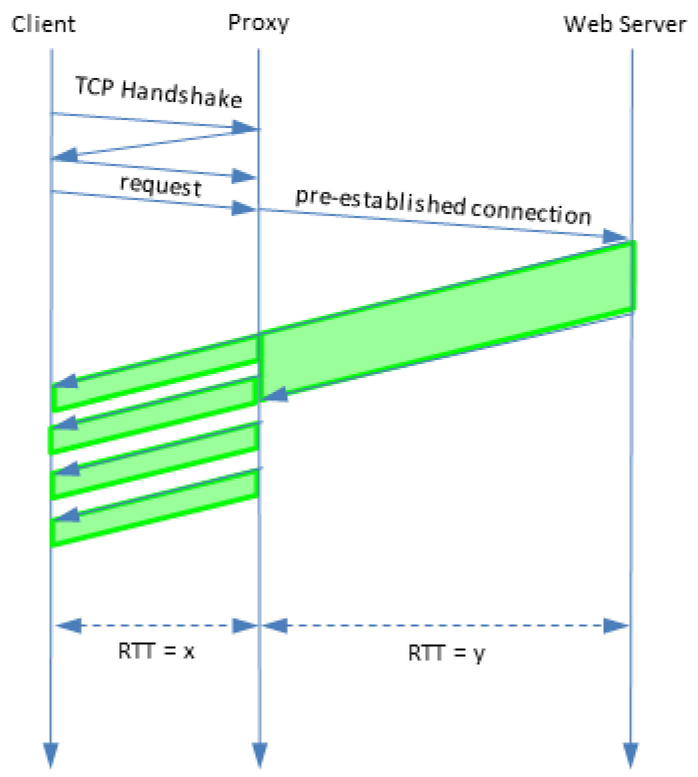


Figure 2.3. Flow diagram of HTTP GET with TCP splitting

of describing some key features and benefits of the newer web browsers. These newer browsers are HTTP 1.1 compliant and have support for multiple simultaneous connections and use of persistent connections with pipelining. Persistent connections reduce latency by cutting down on the RTTs for connection establishment so that successive requests to the same server may be reused. Further the use of multiple connections and pipelining allows more concurrent request to one or more servers and allows for greater parallelism. The impact of these changes is depicted in 2.4. The first flow diagram using HTTP1.0 requests a web page with multiple embedded objects. This requires multiple RTTs to establish the connection for each of the embedded objects. In HTTP 1.0 compliant browsers this is done because the connections are non-persistent. For each additional object requested from a server these RTTs are incurred adding latency. The following flows illustrate a single connection that is persistent and then a flow that utilizes pipelining with a persistent connection. Using persistent connections this reduces the number RTTs for connection establishment and the ability to issue multiple requests with pipelining reduces the delay of queued requests. With the increasing number of embedded objects being included on web pages [38] these features are key to reducing overall latency. Firefox 5.x made a change in the way they reuse persistent connections by sorting and reusing the connection with the largest sending congestion window on the server. Previous versions of Firefox and other browsers would use a FIFO (first in first out) model for reusing persistent connections to a server. This more intelligent scheme used in Firefox 5.x and later Mozilla browsers reduces latency by potentially lessening the number of packets sent to complete a request. The distinction between connections to the same server based on their sending congestion window size means that fewer RTTs may be necessary to retrieve embedded objects. Consider a request for a 100 Kb embedded object from the server with multiple persistent connections. A connection that transmitted more data will have grown its sending congestion window according to the TCP slow-start congestion control strategy and request the embedded object in a single RTT. Compare that to a connection that may have been established earlier but has transmitted fewer packets and less data. The congestion window size of this connection may require multiple RTTs to transmit the same embedded object and introduce additional latency. By making more intelligent decisions regarding persistent HTTP connections Firefox avoids some of the slow start characteristics of TCP. Another feature of newer web browsers is the support for prefetching. Mozilla Firefox and Google Chrome browsers have some prefetching capabilities. These prefetching capabilities are solely directed by the web page being viewed through the use of link tags. Given these link tags in a web page both Firefox and Chrome will retrieve the link in the background. A key difference in Chrome is that it will "pre-render" the page. Pre-rendering is not just downloading the page but also running all the associated JavaScript and rendering all the embedded resources whereas Firefox only downloads the link to the cache. Firefox does not support prefetching of query string URLs and versions prior to 3.5 do not support prefetching of HTTPS links. The newer Google Chrome browser, with its "Google Instant Features", does prefetching and pre-rendering of Google search page results. This is done on the top search results and only on what it deems highly likely followed links. An improved compression protocol, Shared Dictionary Compression over HTTP (SDCH) [37], was also introduced by Google Chrome to reduce transmission delay. This protocol is based on a common dictionary file and pages are automatically built from pieces of the dictionary. When considering shared pieces of HTML documents like templates, CSS style sheets, and JavaScript, this could have significant impact on the amount of repetitive content sent over a connection. Because of the emphasis developers of web browsers are placing on even milliseconds of response time to gain a competitive edge, there is value in the browsing application having this information and the ability to react to it.

The extension developed for this thesis is only applicable for the Firefox browser. For this reason it is relevant to mention some of the default settings of the Firefox browser and the ability to modify them. When using persistent connections the default number per server is 6 and the default max connections per host is 15. Although Firefox is HTTP 1.1 compliant it does not enable pipelining by default and when enabled is defaulted to 4 requests per connection. To alter Firefox configurations the about:config URL is used within the browser to access settings and make modifications.

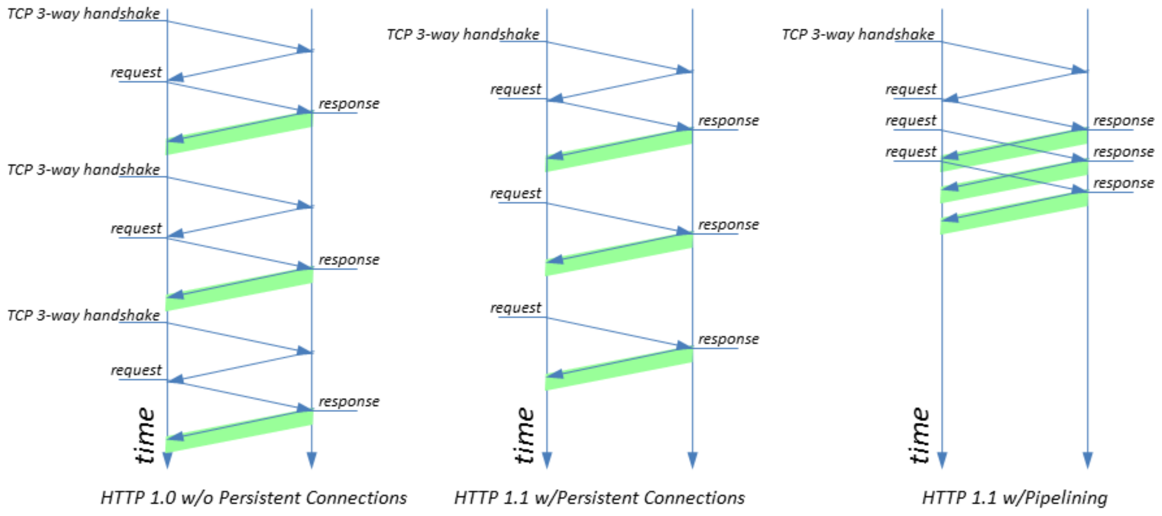

TCP 3-way handshake
request
response
time
HTTP 1.0 w/o Persistent Connections
HTTP 1.1 w/Persistent Connections
HTTP 1.1 w/Pipelining

Figure 2.4. HTTP Connection Improvements

2.4 Broadband Access and Metrics

Caching and prefetching mask the spatial aspect of latency by getting the data closer to the user before it is requested. Without this mask the average US broadband user can expect on average latency to be 41 ms for intra-continental data [10]. This is a single round trip time (RTT) estimate. In some of the longest intra-continental connections this can climb to 134 ms from LA to NY [12]. Compare this to a cellular network and the latency of a 3G or 4G network the average latency can vary widely depending on your carrier and geographic location. To place some solid numbers to latency in 3G and 4G networks consider 2.2 which reflects the findings from OpenSignalMaps 2011 report [39]. These finding show that a 3G network can have latencies ranging from 618 ms to 1114 ms and in a 4G network from 204 ms to 1077 ms. According to Google, the typical load time of a web page on a 3G and 4G network is nine seconds [40]. Current Web workload behavior involving web search and browsing loses its utility under these latencies.

Table 2.2. OpenSignalMaps 3G/4G Network Report

Provider	latency (ms)
ATT 3G	1114 ms
Sprint 3G	618 ms
Sprint 4G	502 ms
T-Mobile 3G	1050 ms
T-Mobile 4G	1077 ms
Verizon 3G	721 ms
Verizon 4G	204 ms

Chapter 3

Design and Implementation

This Section covers the design, architecture, implementation, and use of the WVM extension. It is important to understand the latency factors affecting web browsing so that a discussion of the design choices and their tradeoffs can be presented here along with their impact on the extension. The Design and Implementation section is organized as follows: Section 3.1 covers the design goals, Section 3.2 describes the Firefox extensionn architecture, and Section 3.3 presents the implementation of the WVM Firefox extension along with relevant code and its function. Section 3.4 covers the operation and use of the extension.

3.1 Design Goals

The WVM Firefox extension is a continuation of previous work done on a Firefox 2.0 extension. The goals set forth in this iteration were to not only make the extension supported by the newer Firefox browsers but also make it easier to support in future releases. The new extension needed to be optimized for performance, it needed to be non-blocking of the user interface, it needed to have low impact on the network, it needed to provide the user full control of preferences that affect the extensions behavior, and it needed to provide a framework to expand prefetching capabilities in future work.

This project is a continuation of the work completed for the Firefox extension prototype WVM version 1.0. This project was based both on initial work of Sterbenz et. al., 2002 [9] via the WVM application possessing the following properties regarding feedback to users:

- How long it took for content to be retrieved.
- Qualitative rankings of links on a web page to reflect content retrieval latency.
- Color coding of the links to reflect latency measurements.

The original WVM Firefox extension was designed for environments with low bandwidth, episodic connectivity, and lossy connection characteristics as observed with mobile and wireless users in poor Internet conditions. The initial prototype was restricted in its capabilities. The prototype could only be operated on a Firefox browser version 2.0 to 3.0. The latency checker in the prototype would use Java as an external engine to time the transfer of a set amount of data from the source to the browser. In doing this the prototype would compute a latency score that was a combination of bandwidth and latency for the connection and not true latency. Moving forward, this iteration of the WVM extension needed to:

- Function in newer releases of the Firefox browser.
- Process links in a non-blocking manner.

- Provide the user control of preferences to model the behavior of the extension.
- Optimize the performance of the latency checker for previously checked links.
- Give insight into the cache and the ability to leverage its existence.
- Provide a framework to expand prefetching capabilities.

The benefits of these improvements will result in increased user satisfaction within the application. Users should be advised of potential long delays and by providing this feedback will in affect increase the perceived utility of the application by the user [1] [4].

3.2 Architecture

The architecture of the WVM Latency extension is derived from the Firefox extension technologies. Firefox extensions are used to modify or augment the behavior of the Firefox web browser application. This framework provides mechanisms for extending the user interface, manipulating the Document Object Model (DOM), interacting with the Cross-Platform Component Object Model (XPCOM), and the use of JavaScript to connect them all together.

The previous version of the Firefox extension used the LinkChecker [41] extension as a basis for the WVM extension. The LinkChecker extension already had the capability of color coding links but did so based on its availability and not based on its latency or cache disposition. The WVM Latency extension color coded hyperlinks but did this based on the speed at which an external Java program would download four bytes of data from the targeted hyperlink. A disadvantage of using an external Java program meant that a special mechanism had to be used to pass information from within the Firefox extension to the external Java program and back. These Java calls were made available through custom class loaders. Special security exceptions were also needed to allow the external calls. Using this method to perform the necessary network calls made the extension difficult to port to newer versions of Firefox. Also, because the external Java program returned only a single numeric value reflecting the time in milliseconds to complete the download, useful status and header information from the retrieved hyperlink were ignored. Other difficulties with the early WVM extension were that it had very limited preferences to control the extension's behavior and it could block the user interface if too many links were checked at once. The extension was only supported in Firefox 2.0 and it did not support optimization for previously checked links or domains.

To overcome these issues and also to address new design goals the extension had to be rewritten extensively and changes made to the architecture. Developing the WVM Latency checker as a Firefox extension still seemed to be the best choice for modifying the Firefox web browser to provide latency feedback and user directed controls. The new architecture is founded on the core Firefox technologies: XUL(XML User-interface Language) [42], JavaScript [43], CSS (Cascading Style Sheets) [44], and the XPCOM [45] API for access to lower level functions inside the web browser. Figure 3.1 shows the relationship of these components and how they fit in the overall Firefox browser. The extension uses XUL to build interfaces and CSS styles to change the color and appearance of links. XPCOM is used for accessing the lower level networking functions, to create web workers [46] to provide threading for a non-blocking user interface, and access to the DOM (document object model) to manipulate the web browser. JavaScript functions

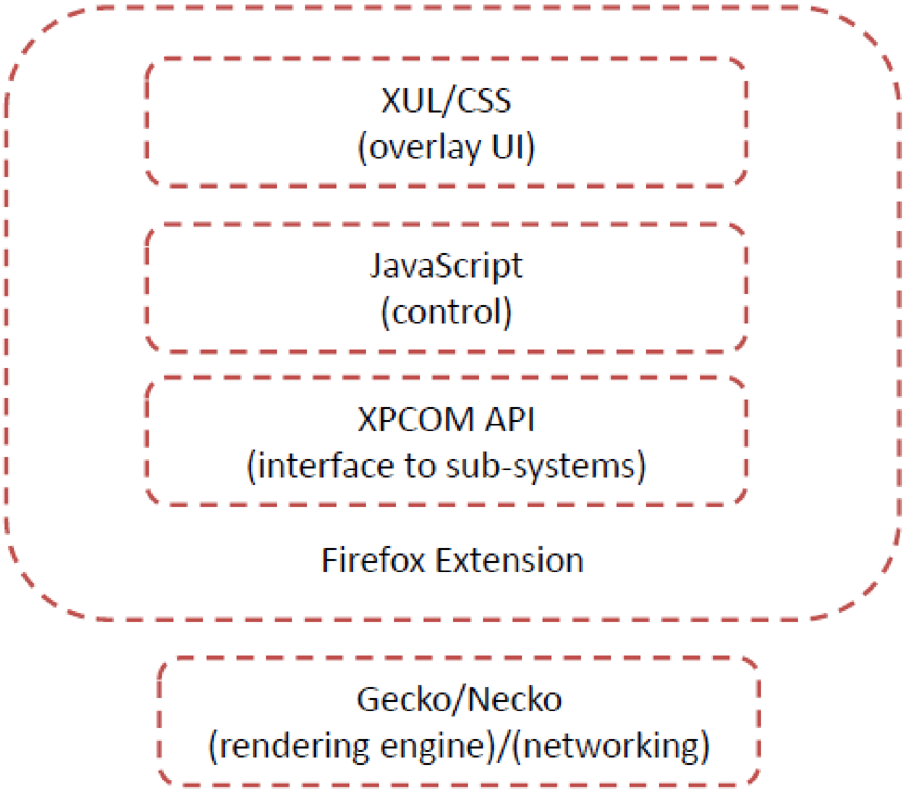


Figure 3.1. Add-on Components

control the access to these pieces and uses JavaScript Code Modules to share data between multiple tabs and browser windows.

The new version of the extension was first updated to work with Firefox version 3.x and then later redesigned again to support version 4.x and above. The redesign of the extension was required because of the thread management change that was made in the layout engine, Gecko [47]. Moving forward with the development these two designs had to be maintained in parallel to support the layout engine change and provide for backward compatibility. Using this architecture the extension could be modified to work with other browsers that utilize the Gecko layout engine. As a future work it may be beneficial to port the extension to a Gecko based browser that operates on a mobile device like a smartphone or tablet.

3.2.1 CSS (Cascading Style Sheets)

Style sheets describe the presentation and attributes of objects on a document in the web browser. When using CSS in a Firefox extension it can modify the format and appearance of the application's user interface and assign custom latency attributes to the objects. Under normal operation the WVM latency extension checks a hyperlink and uses style properties to add foreground and background color, assign latency values, header response status, and if available the data size of the linked object. Using style properties instead of rewriting the web page document on the fly allows the extension to easily toggle the styles on or off as necessary.

Table 3.1. Set Link Attributes

```
// Set custom attributes to store link state information
// Alter the background link color and foreground text color
pageLink.setAttribute("wvmcheck", true);
pageLink.setAttribute("wvmmsg", linkstatusmessage);
pageLink.setAttribute("wvmlatency", gWvmStats.linkLatency[key]);
pageLink.style.color = linkTxtColor;
pageLink.style.background = linkTxtColorBg;
```

The sample code in Table 3.1 demonstrates how a style of the `pageLink` object is modified using the method `setAttribute` and the property of `style`. The `setAttribute` method can either update a current attribute's value or create and set the value of a new attribute if it does not exist. Later in the code the style of the link is modified with a new foreground and background color using the CSS object notation and the values defined in the `linkTxtColorBg` and `linkTxtColor` variables. The link that is being modified is independent of the style. The link could be an image, ftp, http, https, or embedded object and the style will still apply. The general implementation of the Cascading Style Sheets is to separate the presentation formatting from the content.

3.2.2 JavaScript Code Module (JSM)

JavaScript is the primary language of the Firefox browser and allows communication between the different pieces of the extension. JavaScript Code Modules are at the center of the extension and are used to manipulate the interface, manage threads, and make API calls to the browser to perform various operations including network calls and cache access. The advantage of using JavaScript over a language like Java or C++ is that it makes the possibility of porting the extension to other platforms like Fennic (Firefox mobile browser) more viable. If Java or C++ were used, custom loaders and wrappers would need to be included with the extension to make them available and exchange information between the other components. JavaScript Code Modules (JSMs) allow the sharing of code between different privileged JavaScript scopes. In a tabbed browser each new tab that is spawned creates a new instance of the browser window and a new JavaScript scope. These new tabs can potentially create new instances of an extension with each tracking its own data. Without the use of code modules and shared scopes certain optimizations could not be achieved where knowledge of browser activity and history is concerned. One difficulty with JSMs is that the newer versions of Firefox do not allow them to be passed between threads.

3.2.3 XUL (eXtensible User-interface Language)

The XML user interface markup language (XUL) is a language that defines various user interface elements. Examples of the user interface elements include: windows, menus, toolbars, dialogs, satus bars, keyboard shortcuts, and buttons. Developed by the Mozilla project, XUL works with the Gecko rendering engine. XUL allows developers of extensions to create overlay interfaces that modify or add to the existing Firefox user interface. These overlays are a mixture of XML, HTML, CSS and JavaScript. Any part of the user interface can be modified and custom event handlers associated to them. The base XUL interface for Firefox is defined by `browser.xul`. The XUL language also provides mechanisms for accessing the preference engine in Firefox. With access to the preference engine persistent user defined settings can be associated with an extension.

3.2.4 XPCOM (Cross-Platform Component Object Model)

XPCOM is the object model that grants access to virtually all of the underlying functionality of the Firefox browser. Through the use of XPCOM the extension is able to connect to components of the browser like networking, security, thread management, the local file system, and the document object model (DOM). Access to the XPCOM component object model is done through an interface called XPConnect, which provides JavaScript access to the API. An example of an XPCOM interface that could be used in an extension is the `XMLHttpRequest` interface. This interface allows the extension to monitor networking events as an object is retrieved by the browser. Using XPConnect is also how JavaScript interacts with the document that is loaded in the web browser via the DOM object. The use of this interface is how an extension could perform an action such as retrieve all the links on the web page.

3.3 Implementation

The WVM Latency extension is built using the Firefox extension framework. During the development of the extension Mozilla changed their release process and started to use the rapid development model. In the old development model Mozilla would release a new major version once a year. Today Mozilla is releasing a new version of Firefox every 6 weeks. Table 3.2 shows the history of Firefox release since 3.5. This rapid release schedule added a degree of complexity to the

Table 3.2. Firefox Release History

Version	Release Date	Rendering Engine
Firefox 3.5	Jun 30, 2009	Gecko 1.9.1
Firefox 3.6	Jan 21, 2010	Gecko 1.9.2
Firefox 4	Mar 22, 2011	Gecko 2.0
Firefox 5	Jun 21, 2011	Gecko 5.0
Firefox 6	Aug 16, 2011	Gecko 6.0
Firefox 7	Sept 27, 2011	Gecko 7.0
Firefox 8	Nov 8, 2011	Gecko 8.0
Firefox 9	Dec 20, 2011	Gecko 9.0
Firefox 10	Jan 31, 2012	Gecko 10.0
Firefox 11	Mar 13, 2012	Gecko 11.0
Firefox 12	Apr 24, 2012	Gecko 12.0
Firefox 13	Jun 5, 2012	Gecko 13.0
Firefox 14	Jul 17, 2012	Gecko 14.0

overall project due to the testing required. With each updated release of Firefox the extension was tested for compatibility and changes made to the extension's code where needed. A significant change was made at version 4.x and the use of the newer Gecko rendering engine 2.0. This change was so significant that moving forward two clear code bases needed to be maintained. One code base to support Gecko versions 1.9.x and another to support those based on 2.x and later versions. In terms of Firefox releases these code bases were for Firefox 3.6.x and older and Firefox 4.x and later. Currently Firefox 3.6.x has an end of life (EOL) predicted to be sometime in February/March of 2012. Support for new releases of Firefox is based on a 54 week cycle with 9 Extended Support Releases (ESRs). Figure 3.2 visualizes how this release and support cycle would look for Firefox 10. A key difference between the code bases addresses a security feature

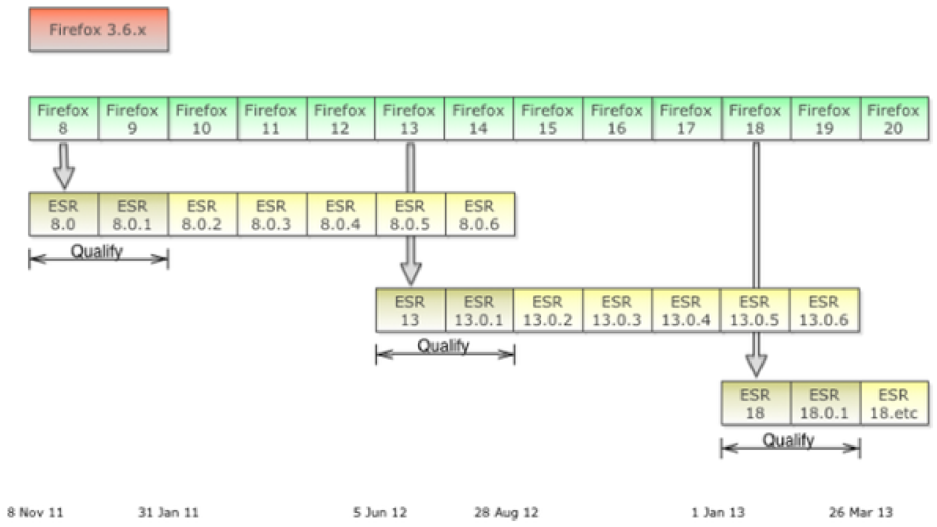


Figure 3.2. Firefox Rapid Release Support

that was implemented in newer releases of Firefox. This security feature had an impact on the threading behavior of the extension. In Firefox version 4.x and later the threads were no longer allowed access to the component interface which the extension used to monitor network events. The extension code that was based on Firefox version 3.x required the use of the component interface in its threads to manage the networking functions and updated shared JavaScript objects for tracking. The use of threading was necessary to avoid blocking of the user interface when performing large batch latency checking. Large batch latency checking can occur when processing links from a search page results set.

Table 3.3 displays the various releases of Firefox along with significant changes that had impact on the WVM addon.

Table 3.3. Firefox Changes by Release

Version	**Changes/Features**
Firefox 3	Introduction of the Thread Manager
Firefox 3.5	Enhanced JavaScript Engine for faster rendering
	Support for WebWorkers
	Prefetching (directed)
Firefox 3.6	Support for HTTP activity observer
	Access to XPCOM from threads
Firefox 4	Support for Chromeworkers
	Disabled the ability to pass JavaScript objects between threads
	Added the ability to use JavaScript code modules in ChromeWorkers
	Introduction of the Web Console
Firefox 5	Reuse of persistent connections by congestion window size
Firefox 8	Support for using XPCOM from ChromeWorkers is removed
	nsIWorkerFactory was removed

Upgrading the WVM Latency extension to support the newer Firefox browsers also meant it had to support the new tabbed browser behavior. The concept of a 'tab', allowed a single browser instance to effectively spawn multiple browsers in a single window. To correctly track and update the hyperlinks being checked by the extension, it needed to monitor all the events that were being triggered by these tabs as the user navigated. This meant activities such as opening or closing a window, adding or removing tabs, navigating to a new web page, and bringing focusing to a window or tab had to be accounted for. The lori (life-of-request info) extension [48] was researched as a model to track such events. Using custom event handlers to monitor browser transitions the WVM Latency extension could then accurately pass information to update the user interface. Tracking data between the tabs or other browser windows for optimization now became much more difficult. Each window or tab now created a new instance of the WVM latency checker and optimization would only be done on a per tab basis and not across the entire browsing session. This greatly reduced the optimization's effectiveness. To combat this, shared objects within the WVM latency extension was created using JavaScript code module (JSMs). JSMs were introduced in Firefox 3.0 and allow the sharing of objects between different privileged scopes. JSMs create a singleton that can then be loaded into a specific JavaScript scope. Information about previously checked links can then be stored in this JSM objects in this scope and shared across all the windows and tabs of the browsing session. Using the Firefox extension framework made development of the WVM latency checker flexible and extensible. Firefox extensions could build upon existing browser functions and interfaces to enhance its functionality. Modifications to the user interface were accomplished with XUL overlays. Menu, toolbar, status bar, and other user interfaces could be built and modified in this manner. These overlays to the user interface could then call the extension's custom functions that were built from JavaScript and XPCOM components. These powerful tools were used to create objects to manipulate networking, cache, preferences, prefetching, and logging.

3.3.1 Threading

Extensions built for Firefox execute on the main thread of the application. If the extension makes too many calls or is not operating in an asynchronous manner blocking of the user interface can occur. The WVM extension, if tasked to processes an excessive number of links could block the user interface and degrade the user experience. To avoid blocking of the user interface the WVM extension employed threading mechanism in the browser to process the links and make network calls. In Section 3.3, a discussion of the implementation briefly mentioned the threading capabilities available in Firefox 3.x and then capabilities available in later version. These different methods for threading are now discussed in greater detail. The thread manager was introduced in Firefox 3. This mechanism was an ideal choice to manage and monitor background jobs that were responsible for networking and latency estimation. The thread manager allowed access to XP-COM objects and component interfaces necessary to perform network requests and monitoring. During normal operation the extension would dispatch a link to the thread to perform a latency estimate. The thread would create an HTTP channel and perform a HEAD request for the link. The estimates were computed through a custom listener attached to the HTTP channel that observed the transitions of the network connection and updated the timers associated to them. To support Firefox 4 and later browsers the extension had to modify its threading behavior. Although the thread manager interface still exists in these browsers, it no longer allowed it to pass JavaScript objects between threads. Shared JavaScript object are a key element of the extension and made the use of the thread manager of no benefit if it could not access them. An alternative to the thread manager was introduced in Firefox 4, ChromeWorkers. ChromeWorkers are a special type of WebWorker that can be used from within JavaScript code modules and have access to XPCOM and XPConnect. ChomeWorkers are background tasks that can perform computationally intense functions and send messages back to the main thread when complete using a callback method. There are several tradeoffs when using ChromeWorkers in place of the thread manager. Most notably is that ChromeWorkers have a large overhead associated with them in terms of memory and are intended to be long-lived. Also ChromeWorkers cannot directly modified the DOM object to updated the user interface.

3.3.2 Networking

The original WVM extension used an external Java applet to make the network calls necessary to compute the latency of links [9]. This estimate was the result of following the link and timing how long it took to download four kilobytes of data. The extension would use this value as its metric for determining a link's latency. This process for computing the link latency would produce consistent results but was more of a bandwidth estimate than a latency estimate. This external Java applet also did not pass back useful header information. To more precisely compute the latency of the link an interface to monitor the network state transitions is needed. The interface needs to be able to identify the various TCP states that occur during an HTTP transaction. Figure 3.3 is representative of a typical HTTP connection to retrieve a document from a web server. The figure includes the associated TCP 3-way handshake and transfer of the document. This figure assumes the document has no associated embedded objects but for the purpose of computing latency of the link it will not impact the calculation. During the 3-way handshake small synchronization packets are exchanged to establish the connection. Ideally if the extension could monitor the sending and receiving of these SYN and SYN/ACK packets the round trip latency (RTL) can be determined. Both `XMLHttpRequest` and `nsIHttpChannel` provide an interface to monitor these state changes in the HTTP connection.

Table 3.4. onReadyStateChange States

State	Description
0	UNINITIALIZED open() has not been called yet.
1	LOADING open() has been called.
2	LOADED send() has been called, headers and status are available.
3	INTERACTIVE Downloading, responseText holds the partial data.
4	COMPLETED Finished with all operations.

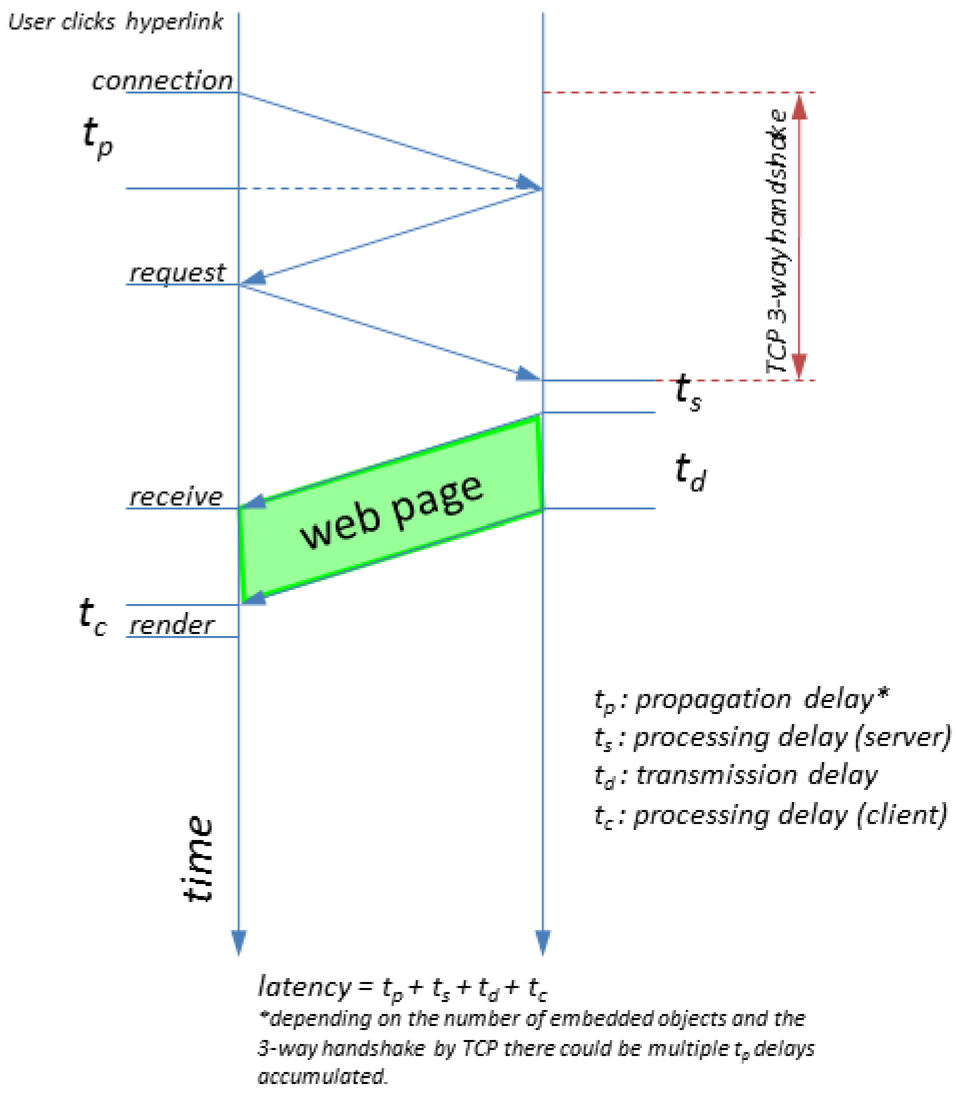

User clicks hyperlink
connection
t_p
request
TCP 3-way handshake
t_s
t_d
web page
receive
t_c
render
time
t_p : propagation delay*
t_s : processing delay (server)
t_d : transmission delay
t_c : processing delay (client)
latency = $t_p + t_s + t_d + t_c$
*depending on the number of embedded objects and the 3-way handshake by TCP there could be multiple t_p delays accumulated.

Figure 3.3. TCP Flow Diagram

To facilitate support for new releases of the Gecko layout engine, the custom Java class that was responsible for computing network latency was replaced with standard XPCOM calls to the `XMLHttpRequest` object. The `XMLHttpRequest` object offers a scriptable interface to build a request and retrieve an object from a URL. Early versions of the extension used this API and its custom event handler, `onReadyStateChange`, to embed the timers for the TCP connection establishment. The states that the XMLHttpRequest object supports are defined in Table 3.4. By placing a start timer in the transition state '1' and a stop timer in the transition state '2' the extension now had true latency estimation.

Table 3.5. nsIHttpChannel States

State	Description
STATUS_RESOLVING	DNS name resolution is being performed.
STATUS_CONNECTING_TO	SYN packet has been sent.
STATUS_CONNECTED_TO	SYN/ACK packet has been received.
STATUS_WAITING_FOR	Waiting for initial response to arrive.
STATUS_RECEIVING_FROM	Data packets have arrived.
STATUS_REDIRECTING	Redirection request received.

The `nsIHttpChannel` object provides a mechanisms for associating a custom progress listener to the callback interface. This progress listener is then used to time the network events and compute the latency of the request. These event handlers in the listener are much more detailed than the `XMLHttpRequest` object. A list of the significant events monitored by the listener are given in Table 3.5. The custom listener can identify events such as the DNS lookup, TCP 3-way handshake, and data transfer. By placing timers in the `CONNECTING_TO` and `CONNECTED_TO` event handers the RTL of the link can be computed.

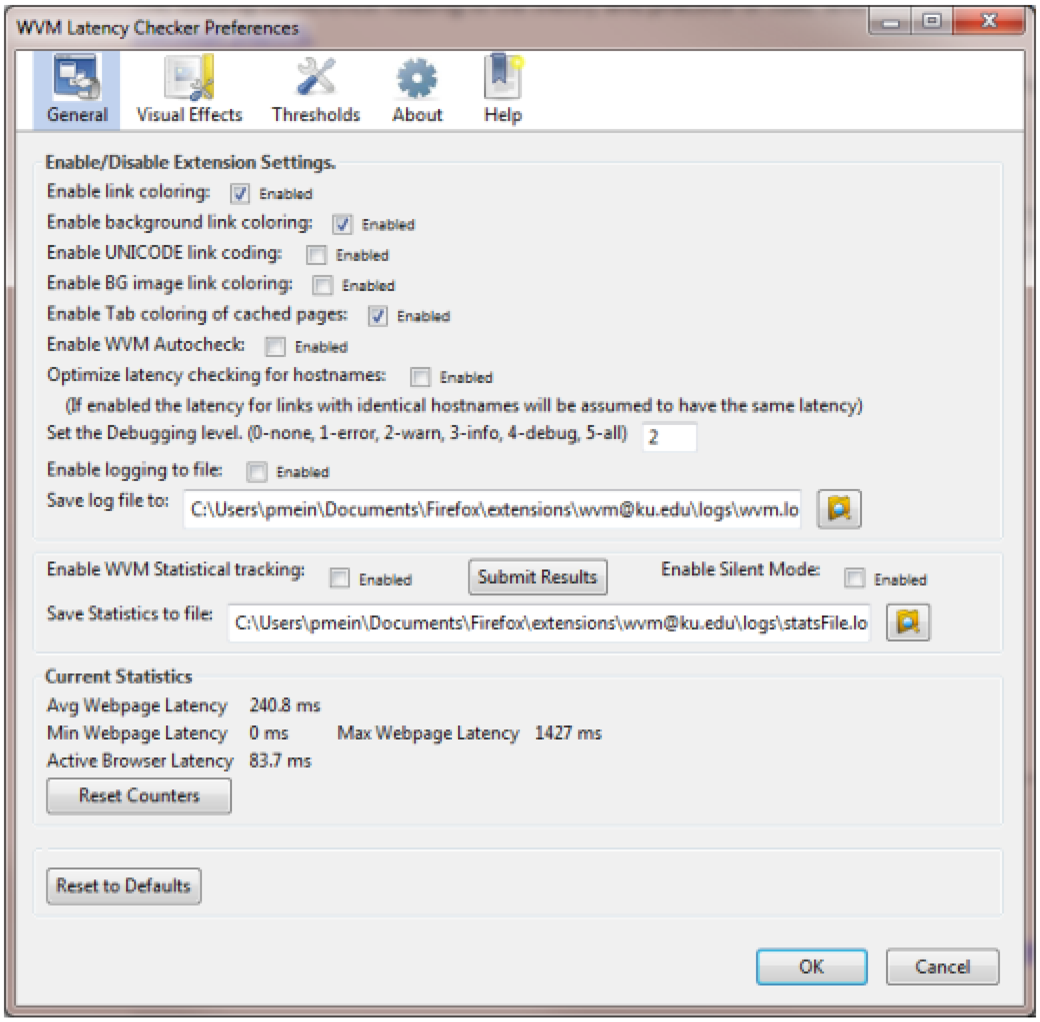
3.4 Operation and Use

To use the WVM latency extension it must first be downloaded and installed in the Firefox browser. Extensions are packaged in XPI format and is easily added or removed through the Firefox extension manager. After installing the extension the browser must be restarted for it to become active. Once the extension is installed it may be modified from its default configuration and used through both toolbar menus and context menus.

The basic functions of the WVM extension include configuring preferences Section 3.4.1, checking links Section 3.4.2, examining the cache 3.4.3 with user directed actions, and pre-fetching documents Section 3.4.4. Each of these functions is described in the following sections. Additionally debugging and logging features are covered in Sections 3.4.5.

3.4.1 Preferences

When the WVM extension 'Preferences' window is opened it will have five different tabs available: General, Visual Effects, Thresholds, About, and Help. Under the 'General' tab are enabling check boxes to turn features on. Table 3.6 list the different features, their default setting, and short description and Figure3.4 . The behaviors listed on the General tab primarily affect the modifications that will be done to the browser web page when the links are checked. Optimize latency checking for hostnames was an enhancement in this version of the extension. Other preferences on the General tab include setting the debugging level and configuring the log file. For troubleshooting and diagnostics the WVM extension can be set to log data to the Firefox error console with varying degrees of granularity. The debugging levels are: 0-none, no data will be logged, 1-error, only error messages



Enable/Disable Extension Settings.

Enable link coloring: [✓ Enabled]

Enable background link coloring: [✓ Enabled]

Enable UNICODE link coding: Enabled

Enable BG image link coloring: Enabled

Enable Tab coloring of cached pages: [✓ Enabled]

Enable WVM Autocheck: Enabled

Optimize latency checking for hostnames: Enabled

(If enabled the latency for links with identical hostnames will be assumed to have the same latency)

Set the Debugging level. (0-none, 1-error, 2-warn, 3-info, 4-debug, 5-all) 2

Enable logging to file: Enabled

Save log file to: C:\Users\pmein\Documents\Firefox\extensions\wvm@ku.edu\logs\wvm.lo

Enable WVM Statistical tracking: Enabled

Enable Silent Mode: Enabled

Save Statistics to file: C:\Users\pmein\Documents\Firefox\extensions\wvm@ku.edu\logs\statsFile.lo

Current Statistics

Avg Webpage Latency 240.8 ms

Min Webpage Latency 0 ms Max Webpage Latency 1427 ms

Active Browser Latency 83.7 ms

Figure 3.4. General tab for WVM preferences

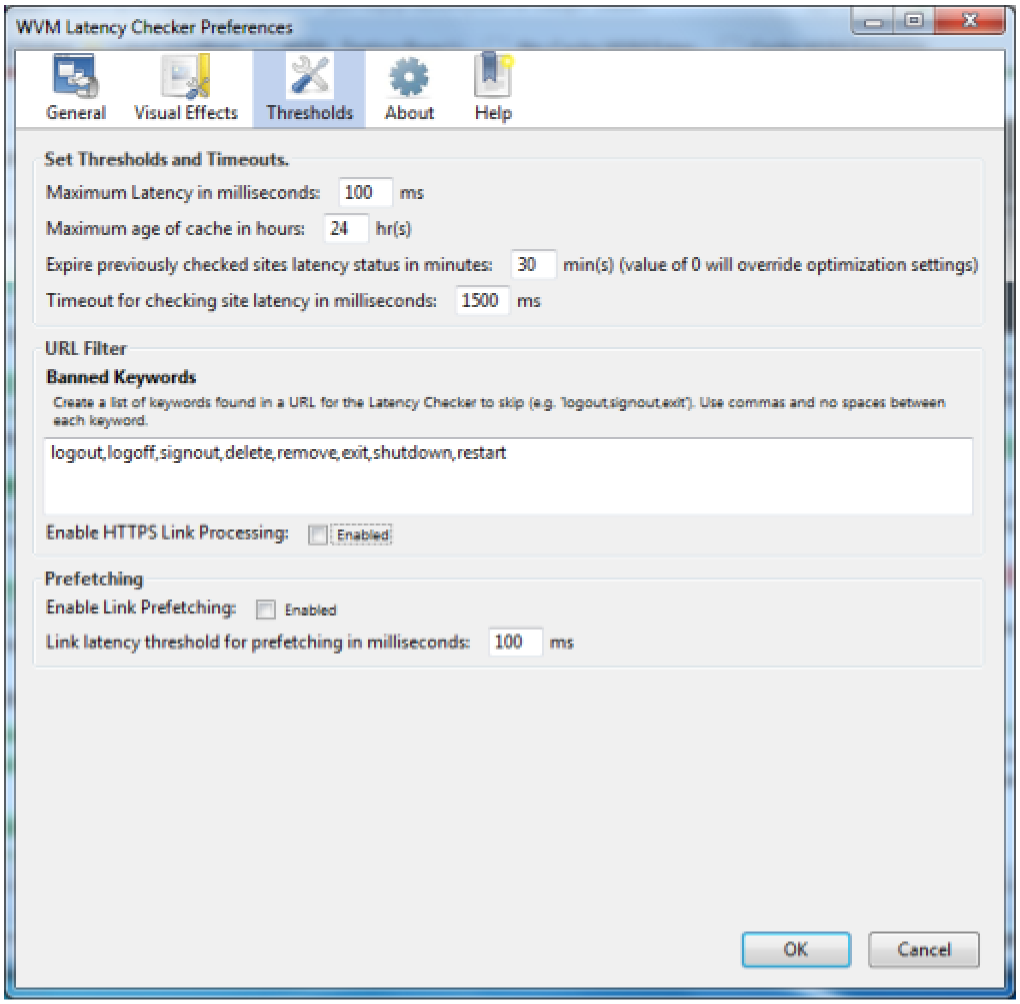
will be logged, 2-warn, error and warning messages will be logged, 3-info, messages at levels 1-2 will be logged as well as informative messages, 4-debug, messages at levels 1-3 will be logged as well as verbose debugging data, 5-all messages will be logged. The default logging level is 2. This logging can be sent to a file and is configurable manually or through a file picker mechanism by clicking the file folder icon. The extension also tracks the overall web page latency estimates and the average browser latency experienced. These are displayed in the 'Current Statistics' box and may be reset through the 'Reset Counters' button. Finally on the General tab an option to restore preferences to their defaults is available by clicking the button ‘Reset to Defaults’.

Table 3.6. General Tab Features

Feature	Description	Default
Enable link coloring	Enables the coloring of links checked by the WVM Latency checker.	Enabled
Enable background link coloring	Enables the background coloring of the link text.	Enabled
Enable UNICODE link coding	Insert a UNICODE symbol in front of the link to denote the latency.	Disabled
Enable BG image link coloring	Places a one pixel wide border around image links to denote their latency.	Disabled
Enable Tab coloring of cached pages	Adjust the color of browser tab if the web page was loaded from the Firefox cache.	Disabled
Enable WVM Autocheck	Enable the WVM extension to automatically check page links when a new web page is opened.	Disabled
Optimize latency checking for hostnames	Skip the checking of links that exist on the same host as a previously checked link.	Disabled
Enable logging to file	Turn logging to a file on and off	Disabled

The 'Visual Effects' tab controls the colors and symbols used when labeling the latency of a link. The first grouping on the tab set the foreground and background colors for each of the different latency levels. Colors maybe set by a hexadecimal RGB color value or by using the color picker immediately to the right of the color value. The next section is the UNICODE symbols defined for the various latencies. Some examples are provided but any valid UNICODE ascii value may be used. Finally the tab color for pages loaded from cache may be set in a similar manner as the link colors.

The 'Thresholds' tab is used for setting timers and filters as well as enabling the prefetch features in the latency extension. Figure 3.5 displays the preferences tab for the threshholds. The Maximum Latency setting is the threshold that determines if a link is determined to be slow or not. This value is in milliseconds. The next setting is the Maximum age of cache and is in hours. This setting determines when a cache is seen as fresh. This value is compared to the difference of the expiration time on the cache entry and the current system time. If the difference is greater than this value the cache is seen as old. Another optimization to this version of the latency extension is the ability to reuse previously calculated latency timers. The WVM extension keeps a list of timestamps associated to the links it has checked. While the latency checker is processing links it will check this list and if the difference between the timestamp and the system time are less than the 'Expire previously checked sites' the link will be skipped and the previous value will be used. The 'Timeout for checking site latency' determines how long the link check process will wait for a response. This value should be higher than the maximum latency threshold and is in milliseconds. The banned keywords list contains a comma separated list of values. If one of these values



Set Thresholds and Timeouts.

Maximum Latency in milliseconds: 100 ms

Maximum age of cache in hours: 24 hr(s)

Expire previously checked sites latency status in minutes: 30 min(s) (value of 0 will override optimization settings)

Timeout for checking site latency in milliseconds: 1500 ms

URL Filter

Banned Keywords

Create a list of keywords found in a URL for the Latency Checker to skip (e.g. 'logout,signout,exit'). Use commas and no spaces between each keyword.

logout,logoff,signout,delete,remove,exit,shutdown,restart

Enable HTTPS Link Processing: ☐ Enabled

Prefetching

Enable Link Prefetching: ☐ Enabled

Link latency threshold for prefetching in milliseconds: 100 ms

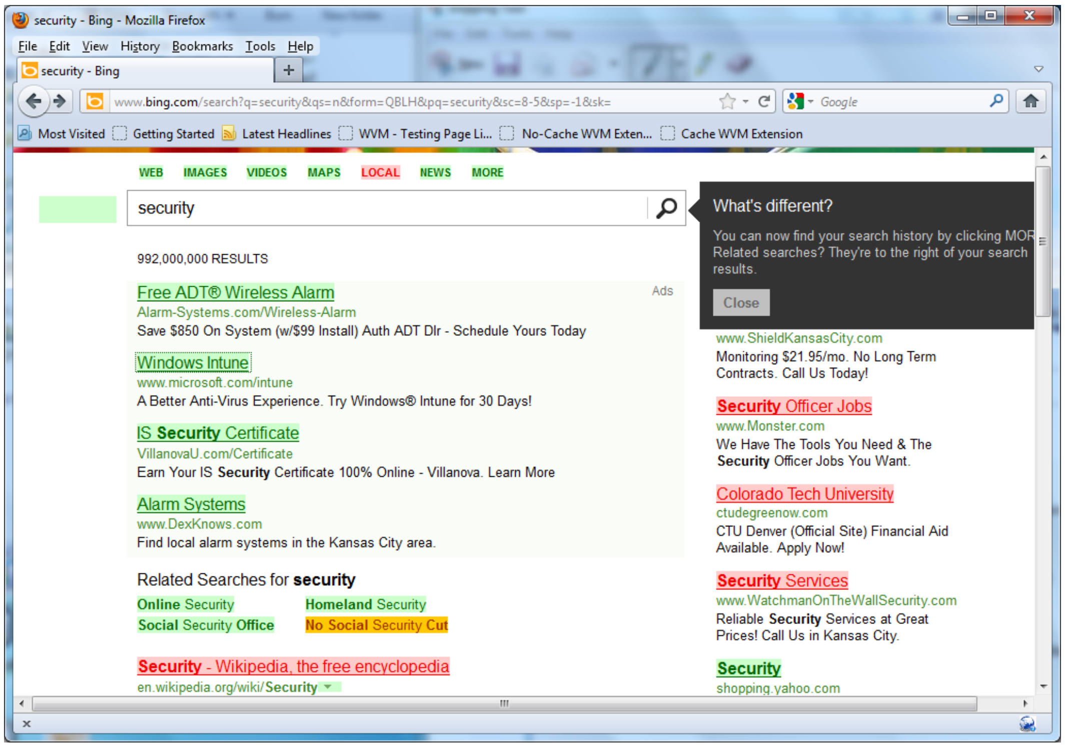
Figure 3.5. Threshholds tab for WVM preferences

is found in the URL of the link being checked the link will be skipped. This helps to skip links that if followed would result in logging out web sessions or other undesirable actions. The last group of preferences enables or disables the prefetching functionality and has a value for the latency of links to be fetched.

3.4.2 Checking Links

There are four different modes for checking hyperlinks with the WVM plug-in: single links, a group of selected links, all links on the page, and auto checking links.

When checking a single link place the mouse over the link and use the right-click action to bring up the context menu options. From the menu select the 'Check Link' option and the WVM plug-in we check the latency of that link. To check a group of selected links, select the area of the web page using the left click and drag capability of the mouse. When the links that need to be check have been selected use the right-click action to bring up the context menu. Choose the 'Check Links in Selected' option and those links will be checked. To check all links on the



What's different?

You can now find your search history by clicking MOR Related searches? They're to the right of your search results.

992,000,000 RESULTS

Free ADT® Wireless Alarm
Alarm-Systems.com/Wireless-Alarm
Save $850 On System (w/$99 Install) Auth ADT Dlr - Schedule Yours Today

Windows Intune
www.microsoft.com/intune
A Better Anti-Virus Experience. Try Windows® Intune for 30 Days!

IS **Security** Certificate
VillanovaU.com/Certificate
Earn Your IS **Security** Certificate 100% Online - Villanova. Learn More

Alarm Systems
www.DexKnows.com
Find local alarm systems in the Kansas City area.

Related Searches for **security**
Online Security
Homeland Security
Social Security **Office**
No Social Security **Cut**

Security - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Security**

www.ShieldKansasCity.com
Monitoring $21.95/mo. No Long Term Contracts. Call Us Today!

Security Officer Jobs
www.Monster.com
We Have The Tools You Need & The **Security** Officer Jobs You Want.

Colorado Tech University
ctudegreenow.com
CTU Denver (Official Site) Financial Aid Available. Apply Now!

Security Services
www.WatchmanOnTheWallSecurity.com
Reliable **Security** Services at Great Prices! Call Us in Kansas City.

Security
shopping.yahoo.com

Figure 3.6. Link coding in the WVM Extension

currently displayed web page either the right-click method for the context menu can be used or go through the Tools menu system to the WVM Latency Checker sub-menu and select 'Check Page Links'. While the WVM plug-in is processing the links the status bar will display its progress Figure 3.7. An example of a web page coded using the WVM extension is shown in Figure 3.6. Links that have been checked may also estimate the documents size if that information was returned as part of the header. The browser status bar will display the estimated latency and the document size if available when the link is hoverd over as in Figure 3.8. Finally the plug-in can be put into Autocheck mode by selecting ‘Enable WVM Autocheck’ from the WVM sub menu. When in Autocheck mode the plug-in will check all hyperlinks on a web page immediately after it has been loaded.

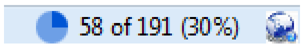


Figure 3.7. Progress status bar

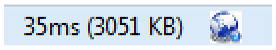


Figure 3.8. Status bar link information

3.4.3 Cache Features

The default behavior of Firefox is to cache a page when it is loaded so that the browser does no need to make additional network requests on subsequent views of the page. If the page is dynamic it may be preferred to fetch the page again to get the latest version. The Firefox cache system determines its behavior while browsing using the `browser.cache.check_doc_frequency` configuration. This configuration has three settings: check for a new version of a page once per session, check for a new version every time a page is loaded, never check for a new version and check for a new version when the page is out of date (default). Retrieving a page from the cache or retrieving a page from the network is done transparent to the user when browsing. There is no indication that the page being viewed is a cached version. The WVM extension monitors the browsers access to the cache and can color code the browser tab to when a page is loaded from the cache to make it explicit when a cache page is being viewed (Enable Tab coloring of cached pages). Depending on the configured behavior of Firefox's document checking frequency older versions of a web page may be pulled from cache when it is more desirable to get a newer version. By color coding the tab it is easily recognizable to the user when a cached page is being viewed.

Displaying cache information of a link can be done through the Firefox hyperlink context menu. When hovering over a link on a web page use the right-click action to bring up the context menu commands (Figure 3.9). If the page refer-

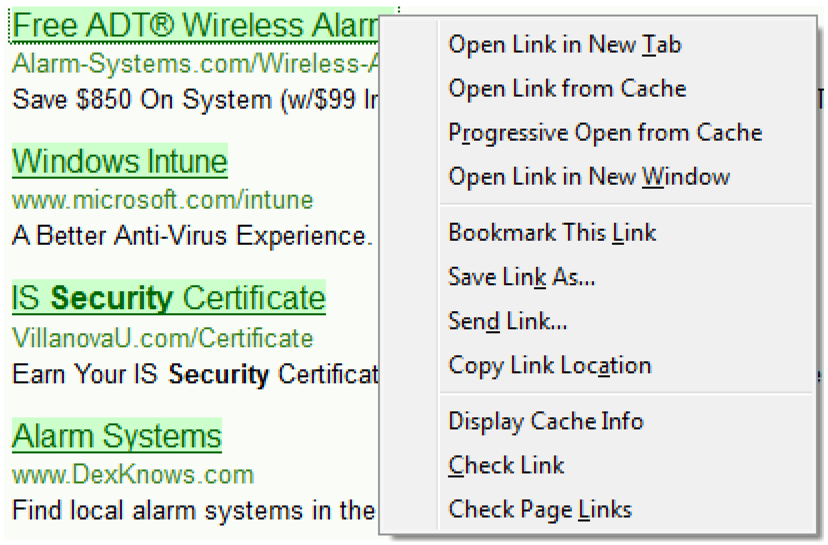


Figure 3.9. Context menu of links

enced by the hyperlink is in the Firefox cache the 'Display Cache Info' option will be available otherwise it will be grayed out as disabled. If the option is available selecting it will retrieve the cache information about the link to include: last fetched, last modified, expires, and data size in bytes. The last fetched date is when the browser last accessed this cached page. The last modified date is the date when the page was last modified. The expires date is when the cache entry should be expired and a new version of the page retrieved when browsing. The data size is the size in bytes of the web page in the cache.

Opening web pages from the Firefox cache using the WVM extension is done through the context menu. When hovering over a link right-click to display the context menu. If the web page is in the Firefox cache alternate opening behaviors will be provided. The first alternate behavior is to ‘Open Link from Cache’. If selected the WVM extension will override the the standard Firefox cache retrieval procedure and forcibly open a cached web page even if it is expired. In situations where a link is under extremely high latency or that a link is down this can be advantageous to the user so that the web page can be viewed. The other alternative behavior is to ‘Progressive Open from Cache’. This behavior will forcibly open the cached version in the browser and create a new tab that will load the current version of the web page from the source. Once the current version has been completely loaded the extension will close the cached version tab and display the current version. The advantage of this behavior is that it allows the cached version to be viewed while a newer copy is loaded in the background.

3.4.4 Link Prefetching

The WVM latency extension provides a framework to support prefetching. As links are checked and latency values determined a threshold can be set in the preferences to prefetch the link if it is highly latent. The prefetching of the link is done by a dedicated web worker responsible for prefetching operations. The prefetching operation is called after a page is loaded and after a link is checked. Theses hooks to the prefetch module allow for future modifications to support both page inference prefetching behavior and link weight prefetching. Different algorithms could be coded into the prefetch module and is a topic discussed as part of the future work. For the purposes of testing and validating the function of prefetching in the extension only a very simple threshold based prefetching scheme has been implemented after a link is checked based on its latency. The linked document is prefetched if it exceeds a user defined threshold. No additional embedded objects associated with the document are retrieved as with pre-rendering. This behavior is similar to the prefetch tag that is supported in Firefox on the server side.

3.4.5 Debugging and Logging

The WVM extension has different debugging levels to log data to the Firefox Error Console and log file. The log level is set through the extensions preferences and the default behavior is 2 - warnings and errors. This will log any warnings or errors the extension encounters with a descriptive messages to the Error Console. To monitor more closely what the extension is doing additional and more detailed logging can be done at levels 3 - info, 4 - debug and 5 - all. Each of these levels displays progressively more and more detail. A sample log entry will contain the timestamp of when the log event occurred, what level the event was triggered at (WARN, INFO, DEBUG, ERROR) and finally which sub-routine the extension was in and finally the message.

Sample log entry:

`1291234213585DEBUG – inCache – http : //www.google.com/webhp?hl = en`

Where:

`1291234213585` - represents the timestamp (UNIX epoch)

`DEBUG` - identifies this is a DEBUG level log message

`inCache` - is the sub-routine in the WVM where the log event took place

`http : //www.google.com/webhp?hl = en` - is the URL be checked in the Firefox cache

Although error and other debugging information is sent to the Firefox Error Console it may be useful to save the log messages from the WVM extension to a file for further inspection. The extension can be configured to send data to a log file. The log data is appended to existing logs so historical data is not lost. This feature is toggled with the ‘Enable logging to file’ and the ‘Save log file to’ preferences.

Chapter 4

Analysis

In this section the accuracy of the latency estimations made by the WVM Latency extension has been evaluated through simulations. These simulations also compare the impact of using the extension in differing Firefox versions and across wired, wireless, and mobile networks. In this section the simulation environment is introduced along with the methodology used to gather the results. Results are then presented with an explanation of their significance.

4.1 Experimental Setup

As stated in earlier sections there are two distinct versions of the WVM Latency extension. These two versions of the WVM Latency extension were evaluated as part of this Thesis. The two versions, 3.6 and 8.0, use specific network and browser functionality that is part of the Gecko rendering engine and Necko networking library. The WVM Latency extension versions support the Firefox browsers that are readily available at the time of this research. Providing a complete test of the extension and take into account the rapid release cylce currently being used by Mozilla multiple versions of the newer Firefox browsers were included in the testing. This was done to determine if there were significant differences in these releases in terms of support and performance. For the testing that follows Firefox browser versions 3.6.38, 6.0.2, and 8.0.1 were used.

To test the WVM Latency extensions across the different Firefox browsers three virtual machines were created. The software used for the virtualization hypervisor was VMWare [49] and their 'VMware Player' software. The guest operating systems were configured with Microsoft's Windows 7 Enterprise 64bit version. The virtual machines were hosted on a Dell Latitude E6400 with 4 gigabytes of RAM. Within the virtual machine the virtual network interface was bridged to the physical adapter. In bridge mode, VMWare uses the physical interface of the host to directly within the virtual machine to emulate its network interface. In this emulation mode it is transparent to the programs running within the virtual machine and the virtual machine appears as another machine on the network environment that the host is connected to. Configuring the simulation in this manner facilitated the testing across multiple network types and Firefox browser versions. The first virtual machine was installed with Firefox 3.6.23 and WVM 3.6. This instance supported the Gecko rendering engine up to version 1.9.2. The other two virtual machines were installed with Firefox 6.0.2 and 8.0.1 and WVM 8.0. These instances supported Gecko version 6.0 and 8.0 respectively.

The network types that were tested included a wired 1 GB corporate network, a public 802.11 wireless network, a stationary tethered 4G cellular network, and finally tethered to a mobile 4G cellular network. These networks represent a diverse use case of web browsing across varying degrees of bandwidth and reliability. For each network type and browser version the WVM Latency extension estimated the

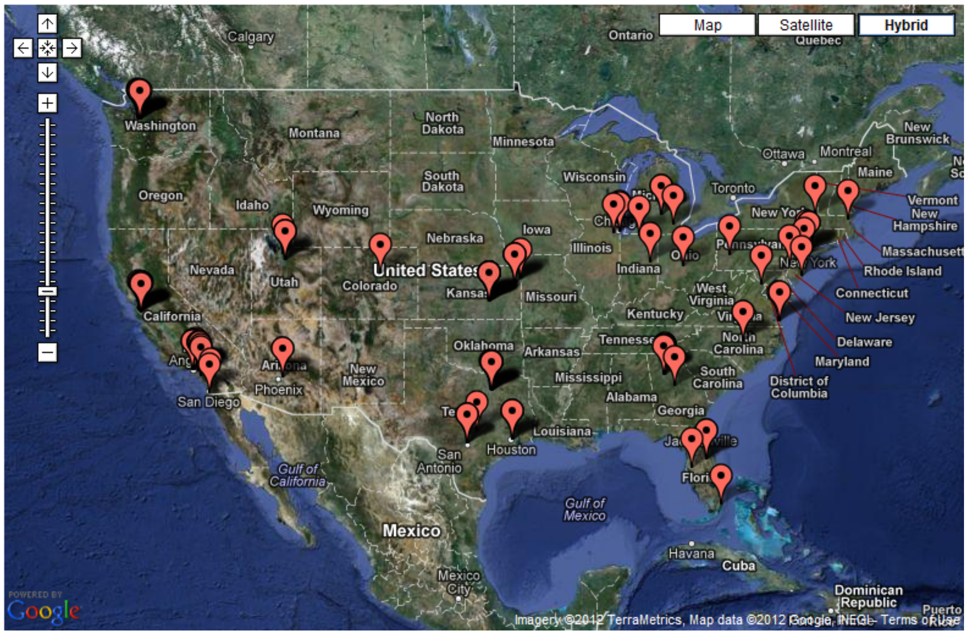


Figure 4.1. Domesic URLs

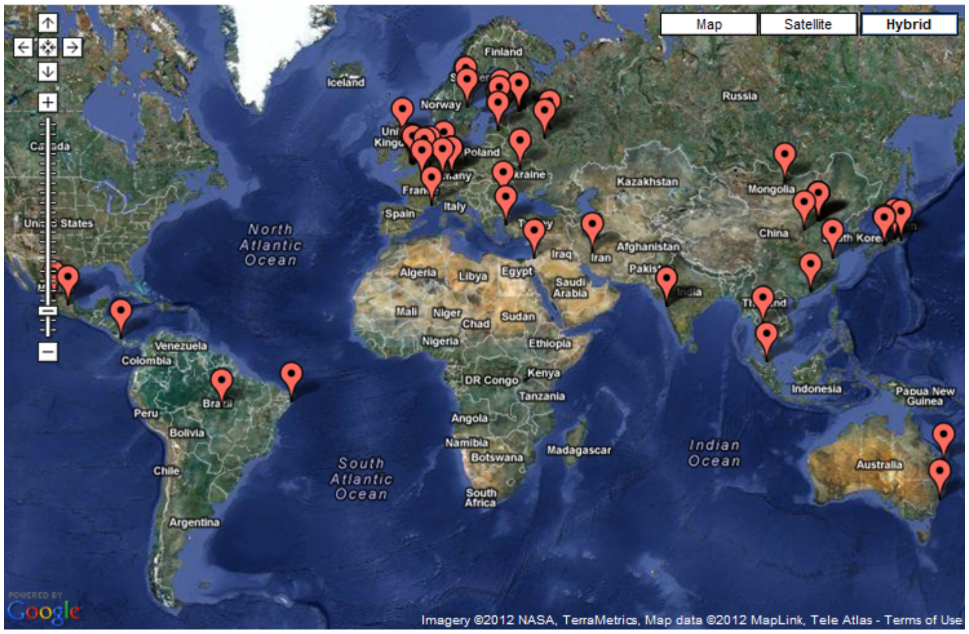


Figure 4.2. Foreign URLs

latency of a 225 URL data set. This data set was generated from a random sampling of 200 URLs from the HTTP Archive [50]. The HTTP Archive is a project that is based off the Internet Archive. The purpose of the HTTP Archive is to record how digitized content is constructed and served. This data can then be used to provide a common data set to conduct web performance research. The archive currently has 264,752 URLs and are based off Alexa Top one million sites [51]. The final 25 URLs were taken from the top 25 URLs of 2011 [51]. These 225 links referenced geographically diverse systems across various top level domains. Figure 4.1 and Figure 4.2 plot the geo-location of the URLs using the publicly available MaxMind [52] GeoIP database. The location of the virtual machines conducting the tests were geographically located in Overland Park Kansas. Table 4.1 and table 4.2 describes the content of the data set showing the distribution of foreign and domestic URLs as well as the top level domains.

4.2 Methodology

To validate how accurately the WVM Latency extension estimated the latency of a link, data had to be gathered to compare the estimates against the true latency of the link. This data needed to demonstrate that the WVM extension would provide repeatable and consistent results across a variety of network types and across multiple versions of the Firefox web browser. It was expected that the WVM Latency extension would estimate the latency of links with little difference from the actual latency experienced when opening the link. It was also expected that networks with less reliability would have an adverse effect on the accuracy of the extension but still provide usable estimates.

To accomplish this quantitative analysis a testing module was built into the

Table 4.1. URLs by Country

Country	Number
United States	172
China	7
Russian Federation	5
United Kingdom	5
Japan	4
Germany	4
France	3
Mexico	2
India	2
Ukraine	2
Australia	2
Sweden	2
Brazil	2
Thailand	1
Israel	1
Iran	1
Belgium	1
Azerbaijan	1
Greece	1
Finland	1
Estonia	1
Malaysia	1
Costa Rica	1
Mongolia	1
Latvia	1
Romania	1

Table 4.2. URL Distribution

Top Level Domain	Number
.com	158
.net	10
.uk	6
.org	6
.in	4
.ru	3
.fr	3
.cn	3
.de	3
.us	2
.mx	2
.edu	2
.br	2
.ua	2
.gr	2
.au	2
other	15

extension. The operation of the testing module included a data file of URLs, discussed in section 4.1, to be checked for latency. This data file was loaded into an array and then processed sequentially using the extension's latency estimator engine. After estimating the latency of the links the testing module would then begin to navigate to each URL in the browser and record the actual browser latency when opening the web page.

Information about the link being checked, preferences of the extension, and the latency estimate were logged to a file for later analysis. A sample of the data file is show in Table 4.3. The data file starts by providing header information describing the location, network type, date, and the platform the testing was performed on. After the header information data entries contain a timestamp, the hostname, URL, latency estimate in ms, actual browser latency in ms, unused data element, and a flag to indicate if domain optimization was enabled to skip previously estimated domains.

Testing was performed by first clearing the cache of the web browser and resetting the WVM Latency extension's preference to their default settings. Logging was then configured for a debugging level. The testing module was then initiated. After a cycle of estimating the links and opening them was complete the browser's cache was cleared and then closed. The testing cycle was repeated five times for each browser version and network type to average the results. After the testing was complete the logs were parsed and the differences of the estimates from the actual browser latency were computed. The aggregated data was separated by browser version and network type to generate complimentary cumulative distribution (CCDF) plots using Gnuplot.

Table 4.3. Testing data file

Timestamp	Entry
1333643349529	Location: JCCC
1333643349529	Network: DSL
1333643507416	Logfile Opened on Thursday, April 05, 2012 11:31:47 AM
1333643507416	Platform: WINNT(Mozilla:8.0.1)
1333643594574	Firefox8 VM,http://technet.microsoft.com/en-us/library/cc778798(WS.10).aspx,89,89, undefined,false
1333643595109	Firefox8 VM,http://systemcentersolutions.wordpress.com/feed,NA,38,undefined,false
1333643595271	Firefox8 VM,http://click.andomedia.com/b/2/91051,115,120,undefined,false
1333644942639	Firefox8 VM,http://facebook.com,41,49,undefined,false
1333644942964	Firefox8 VM,http://www.dafont.com/en/top.php,130,142,undefined,false

4.3 Results and Analysis

The first network that the simulation was performed on was a wired 1 GB local area network. The latency estimate compared to that of the actual browser behavior was expected to differ by only a small amount. This is due to the reliability of the connection and its low congestion. The extension was tested in Firefox browser versions 3.6.38, 6.0.2, and 8.0.1. The results of how much the estimated value differed from the actual latency browsing to the link are shown in the CCDF plot Figure 4.3. Respectively the estimates are accurate within 8 ms, 15 ms, and 18 ms with 90% confidence. Both Firefox 6.0.2 and 8.0.1 share a similar rendering engine and similar core APIs for the networking functions. The tightness of the CCDF plot between these two versions may be attributable to this fact. The WVM Latency extension that was developed for Firefox 3.x is more accurate in its estimates than the later version. The key difference between the extension versions is their thread manager. The use of Web Workers in the Firefox 6.0.2 and 8.0.1 latency extension may be a contributing factor.

Table 4.4. 1GB LAN Latency Estimates

Browser Version	Average Browser Latency ms	Difference in ms at 90% CCDF	Average accuracy at 90% CCDF
Firefox 3.6.38	75	8	96%
Firefox 6.0.2	74	15	93%
Firefox 8.0.1	76	18	93%
Overall	75	15	94%

Table 4.4 gives additional information about the findings. What is of interest in this table is not only the overall accuracy compared to the browsing latency but the accuracy at the 90% CCDF level. Inspecting the overall performance of the extension on a reliable network link such as the wired 1 GB LAN the accuracy

of prediction is 88%. At the 90% CCDF level the extensions is 94% accurate at predicting the latency. From a qualitative view this shows that the WVM Latency extension is extremely reliable at predicting the latency of links and the results are highly reproducible.

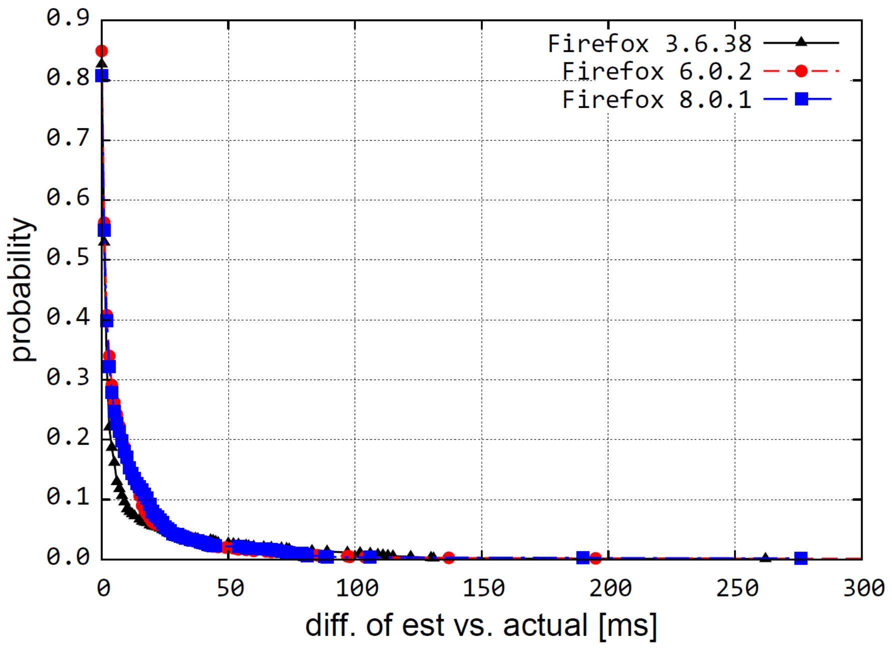


Figure 4.3. Latency Estimates over a 1GB LAN connection

Next less reliable networks were tested. These networks included a wireless public 802.11 network and a 4G cellular tethered network. The difference from the estimated latency to the actual browser latency are show in the CCDF plot Figures 4.4 and 4.5. Compared to the wired 1 GB LAN connection the estimates are less accurate. The accuracy of the estimates across these networks at the 90% confidence are 51 ms for the 802.11 network and 85 ms on the 4G cellular network. Wireless networks are more susceptible to errors, retransmissions, and congestion from shared use. Considering these factors it is not surprising that the accuracy of the extensions has degraded on these networks. What is noticeable however is that again the WVM extension performed better in the previous version of the Firefox browser. On the wireless networks the extension is 74% accurate.

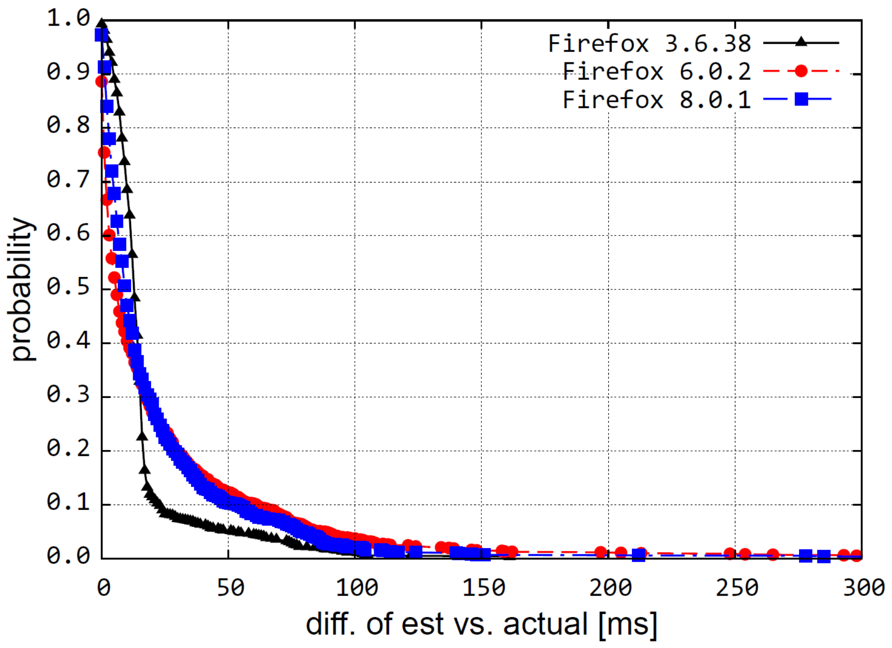


Figure 4.4. Latency Estimates over wireless public 802.11 connection

Finally a 4G cellular network was tested with mobility by tethering the laptop to a smartphone in a moving vehicle Figure 4.6. The overall latency may change due to the proximity to a cell site, the capacity of the cell site, the number of other users connected to the same cell site, the surrounding terrain, and radio frequency interference. The extension performed similar in both Firefox 3.6.38 and 8.0.1 but still not as well as stationary wireless. An overall comparison of the accuracy of the latency extension in Firefox 8.0.1 is given in Figure 4.7.

This figure demonstrates that the accuracy of the extension and that it diminishes in the wired to wireless and mobile platforms. The degree of errors, retransmissions, and congestion associated with these networks matches this pattern and does not take away from the utility of the WVM Latency extension. In

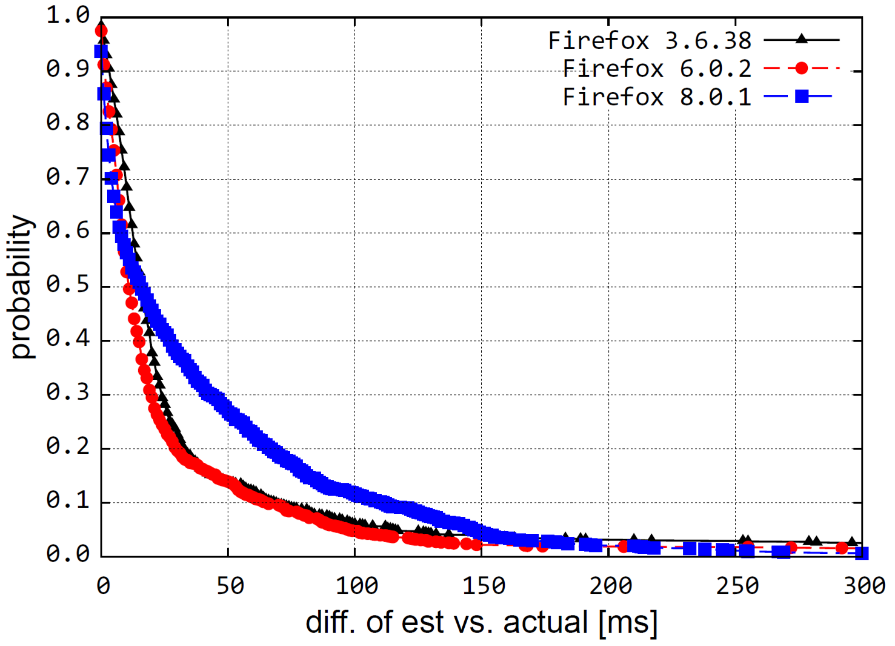


Figure 4.5. Latency Estimates over cellular 4G tethered connection

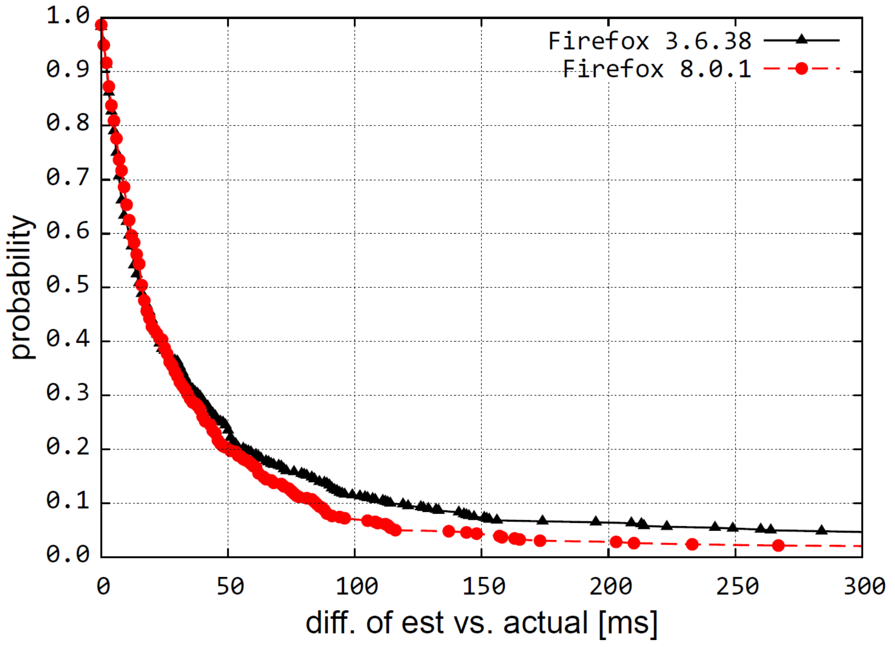


Figure 4.6. Latency Estimates over cellular 4G mobile connection

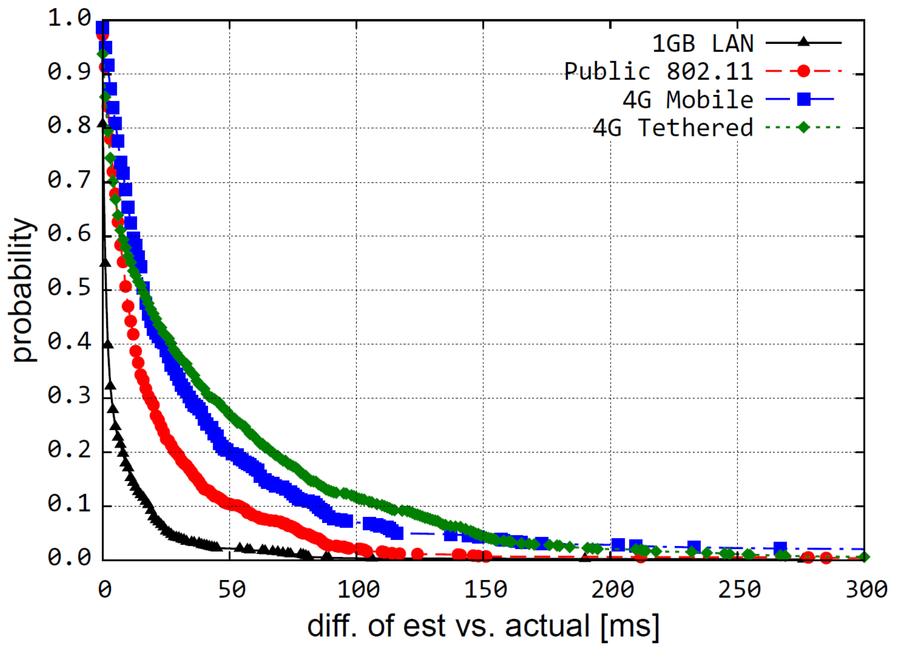
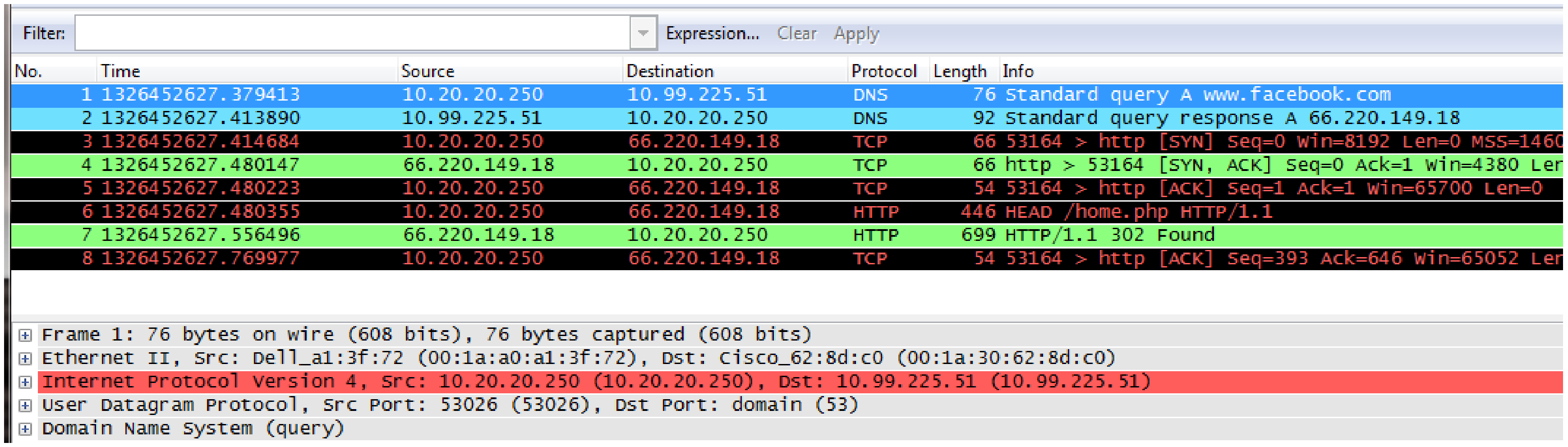


Figure 4.7. Latency Estimates for Firefox 8.0.1 over differing networks

a final attempt to demonstrate the soundness of the latency extension a packet capture was done using Wireshark during normal operation and compared to the debugging log of the WVM Latency extension. This information was used to validate the extension was executing its event handlers correctly during the TCP 3-way handshake. Figure 4.8 and Figure 4.9 show a side-by-side comparison of one such trace.

In the trace of Wireshark the first action shown in Figure 4.8 is the DNS query to resolve the links address. This is shown to happen at a timestamp of `1326452627.379413`. Comparing this to the debugging log's of Figure 4.9 and the entry with status of `STATUS_RESOLVING` and timestamp of `1326452627368` the two logs can be aligned to track their progress through establishing a connection with the remote host. In the Wireshark log at packet number 3 the SYN packet is sent and then the SYN/ACK packet is received 66 ms later at packet number

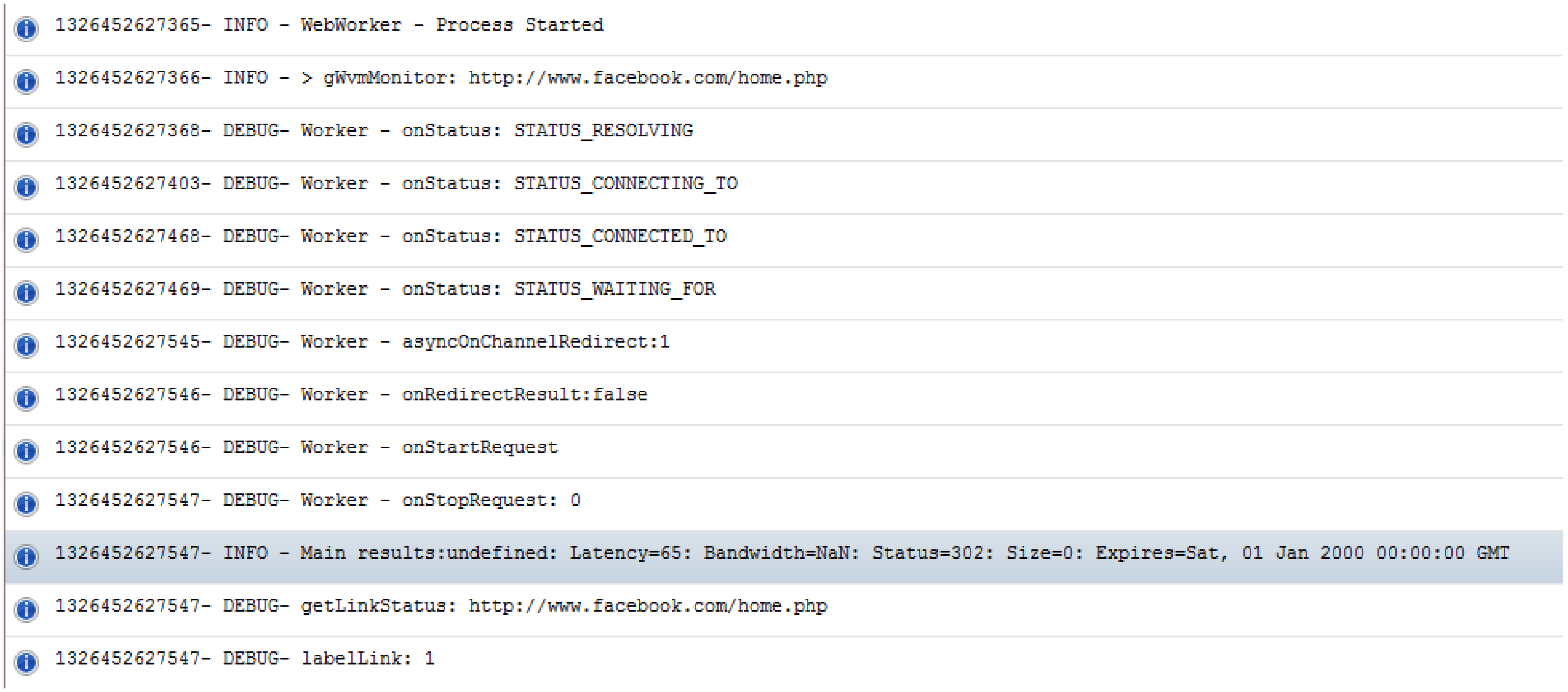
4 of the trace. For the purposes of the WVM Latency extension this is what it uses as its metric for estimating the latency of a link, the single round trip time to a host. Following the debugging log the `STATUS_CONNECTING_TO` represents the ACK at timestamp `1326452627403` and the `STATUS_CONNECTED_TO` represents the SYN/ACK at timestamp `1326452627468`. The estimate by the WVM Latency extension reports that the latency is found to be 65 ms. These results show that at comparable timestamps the packet trace and the extension's log both identify the SYN, SYN/ACK, and ACK packets of the connection establishment and there is only a 1 ms difference between the reported round trip latencies.



```
Filter:                                  Expression...  Clear  Apply
No.   Time                Source          Destination     Protocol Length Info
    1 1326452627.379413   10.20.20.250    10.99.225.51    DNS        76 Standard query A www.facebook.com
    2 1326452627.413890   10.99.225.51    10.20.20.250    DNS        92 Standard query response A 66.220.149.18
    3 1326452627.414684   10.20.20.250    66.220.149.18   TCP        66 53164 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460
    4 1326452627.480147   66.220.149.18   10.20.20.250    TCP        66 http > 53164 [SYN, ACK] Seq=0 Ack=1 Win=4380 Ler
    5 1326452627.480223   10.20.20.250    66.220.149.18   TCP        54 53164 > http [ACK] Seq=1 Ack=1 Win=65700 Len=0
    6 1326452627.480355   10.20.20.250    66.220.149.18   HTTP      446 HEAD /home.php HTTP/1.1
    7 1326452627.556496   66.220.149.18   10.20.20.250    HTTP      699 HTTP/1.1 302 Found
    8 1326452627.769977   10.20.20.250    66.220.149.18   TCP        54 53164 > http [ACK] Seq=393 Ack=646 Win=65052 Ler

⊞ Frame 1: 76 bytes on wire (608 bits), 76 bytes captured (608 bits)
⊞ Ethernet II, Src: Dell_a1:3f:72 (00:1a:a0:a1:3f:72), Dst: Cisco_62:8d:c0 (00:1a:30:62:8d:c0)
⊞ Internet Protocol Version 4, Src: 10.20.20.250 (10.20.20.250), Dst: 10.99.225.51 (10.99.225.51)
⊞ User Datagram Protocol, Src Port: 53026 (53026), Dst Port: domain (53)
⊞ Domain Name System (query)
```

Figure 4.8. Wireshark Packet Trace



```
1326452627365- INFO - WebWorker - Process Started
1326452627366- INFO - > gWvmMonitor: http://www.facebook.com/home.php
1326452627368- DEBUG- Worker - onStatus: STATUS_RESOLVING
1326452627403- DEBUG- Worker - onStatus: STATUS_CONNECTING_TO
1326452627468- DEBUG- Worker - onStatus: STATUS_CONNECTED_TO
1326452627469- DEBUG- Worker - onStatus: STATUS_WAITING_FOR
1326452627545- DEBUG- Worker - asyncOnChannelRedirect:1
1326452627546- DEBUG- Worker - onRedirectResult:false
1326452627546- DEBUG- Worker - onStartRequest
1326452627547- DEBUG- Worker - onStopRequest: 0
1326452627547- INFO - Main results:undefined: Latency=65: Bandwidth=NaN: Status=302: Size=0: Expires=Sat, 01 Jan 2000 00:00:00 GMT
1326452627547- DEBUG- getLinkStatus: http://www.facebook.com/home.php
1326452627547- DEBUG- labelLink: 1
```

Figure 4.9. WVM Debug Log

Chapter 5

Conclusions and Future Work

This thesis has presented a Firefox extension that provides latency aware information to the user so that they may take directed action based on this knowledge. It reviewed the contributors to web application latency and possible remedies, discussed current web browser technologies and explained the design, implementation, and use of the WVM Latency extension. This final chapter reflects on the results and future work. First, Section 5.1 discusses the thesis's main contributions but also mentions its limitations and boundaries. Section 5.2 presents possible future work to be carried out on the foundations of this thesis.

5.1 Contribution

This thesis provided an improvement to the web browser application. The WVM Latency extension added latency aware feedback to the user and a set of tools to take directed action when responding to latency. The following list summarizes the most significant contributions of the WVM Latency extension.

- Analysis of the WVM plug-in accuracy accross browser releases and different networks.

- Latency checking of links with styles applied to change the color or appearance.

- Status bar information of checked links to include latency timer and page size.

- Display the disposition of cached page with expiration, modified, and last fetched time.

- Open link from the source, cache or combination.

- Identify pages retrieved from cache by applying color to web page tab.

- Prefetch links based on latency.

- Customizable preferences to model the behavior of the tool.

The extension also had some additional benefits incurred while checking links. During the latency checking DNS resolution is performed and TCP connections are established. If the link is followed by the user they will benefit from the cached DNS query and the potential use of a persistent TCP connection. These features of the extension will reduce the perceived latency of the web page. The extension does have very simplistic prefetch capabilities but has the framework in place where more robust prefetch algorithms and functions could be placed. With the above contributions there are also some limitations of the extension. The latency checker performs an HTTP HEAD when computing the latency. When a true HTTP GET is done the latency may be significantly higher due to process delay of a dynamic web page. This can lead to some estimates being inaccurate.

The current WVM extension operates on a single worker thread which can be a bottleneck if a large number of links are processed. Lastly, when multiple browser windows are open navigation in an alternate window will halt the processing of a latency check that is underway.

5.2 Conclusion

Does the WVM Latency extension accurately estimate the latency of a link for the user? The quantitative result gathered through the testing clearly demonstrates that the extension does so within 8 - 18 ms with a 90% confidence on a 1 GB wired connection. The estimates were expected to be better on these networks that were less episodic compared to that of wireless 802.11 or 4G cellular networks. This was shown to be the case with the results from the testing. The stationary tethered 4G tests did have acceptable results with latency estimates being 73% accurate and deviating from the actual browsing latency by less than 70 ms. When identifying the sources of errors in the estimates during testing there are the standard issues faced on the network which include congestion and lost or errored packets that must be transmitted. Additional error in the estimate that can be attributed directly to the extension may result from the request method used by the extension. During the estimation process a http HEAD request is sent compared to an http GET during the actual verification of the browsing latency. This http GET results in a full page request. If the request is for a dynamic web page the processing delay on the server may skew the estimate to a lower than actual value. This tradeoff to make a HEAD vs. GET request was done to reduce network overhead from the extension. As stated earlier the extension does provide useful information to the end user. Given that the WVM Latency extension does accurately provided this feedback to the user, along with the tools built into the extension for user directed behavior it can be leveraged to react to high latency links.

5.3 Future Work

The WVM Latency extension provides a framework for prefetching links. The extension as it sits today provides a prefetch mechanism that is based on the latency estimate and a threshold. The prefetch only retrieves the document and does not retrieve embedded objects, style sheets, or JavaScript. Because the prefetch module retrieves the document as a DOM object the module could be modified to also retrieve the embedded object offering not only prefetch capability but pre-rendering capabilities as well. Future versions of the extension could leverage better Predictive Partial Match or Dependency Graph algorithms to produce a more accurate predictive engine for the prefetch module. As seen from the results, wireless and cellular networks suffer more from latency. As smartphones and tablets continue to grow as the preferred device updating the extension to work on mobile platform will also be beneficial. There currently is a single worker thread limit. Providing a user preference to set the thread count to process latency checking would be an advantage for processing large sets of links for search results.

References

[1] John A. Hoxmeier Ph. D and Chris Dicesare Manager. System response time and user satisfaction: An experimental study of browser-based applications. In *Proceedings of the Association of Information Systems Americas Conference*, pages 10–13, 2000.

[2] Walter J. Doherty. The economic value of rapid response time. White Paper, IBM Dept. 824, November 1982.

[3] Inc. Zona Research. The economic impacts of unacceptable web site download speeds. White Paper, Zona Research Inc., April 1999.

[4] Anna Bouch, Allan Kuchinsky, and Nina Bhatti. Quality is in the eye of the beholder: meeting users' requirements for internet quality of service. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '00, pages 297–304, New York, NY, USA, 2000. ACM.

[5] Josep Domènech, Ana Pont, Julio Sahuquillo, and José A. Gil. A user-focused evaluation of web prefetching algorithms. *Comput. Commun.*, 30:2213–2224, July 2007.

[6] Jay Chen, Lakshminarayanan Subramanian, and Kentaro Toyama. Web search and browsing behavior under poor connectivity. In *Proceedings of*

the 27th international conference extended abstracts on Human factors in computing systems, CHI EA 2009, pages 3473–3478, New York, NY, USA, 2009. ACM.

[7] Joe Weinman. As time goes by: The law of cloud response time. Working Paper, April 2011.

[8] Liam Quin. Xul (xml user interface language). `http://www.w3.org/XML/`, January 2012.

[9] James P.G. Sterbenz, Tushar Saxena, and Rajesh Krishnan. Latency-Aware Information Access with User-Directed Fetch Behaviour for Weakly-Connected Mobile Wireless Clients. *BBN Technical Report*, (8340):1–10, 2002.

[10] Verizon ip latency statistics. `http://www.verizonbusiness.com/about/network/latency/`, 2011.

[11] Jordan Nielson, Carey Williamson, and Martin Arlitt. Benchmarking modern web browsers. In *IEEE HotWeb, 2008*, October 2008.

[12] Att latency statistics. `http://ipnetwork.bgtmo.ip.att.net/pws/network_delay.html`, 2011.

[13] Sandvine's spring 2011 global internet phenomena report reveals new internet trends. `http://www.sandvine.com/news/pr_detail.asp?ID=312`, May 2011.

[14] S. Biaz and Shaoen Wu. Loss differentiated rate adaptation in wireless networks. In *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*, pages 1639 –1644, April 2008.

[15] Giuseppe Bianchi, Fabrizio Formisano, and Domenico Giustiniano. 802.11b/g link level measurements for an outdoor wireless campus network. In *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, WOWMOM '06, pages 525–530, Washington, DC, USA, 2006. IEEE Computer Society.

[16] James Sterbenz and Joseph Touch. *High-Speed Netowrking: A Systematic Approach to High-Bandwidth Low-Latency Communication*. Wiley, 2001.

[17] F5. Increase ssl offload performance with big-ip. `http://www.f5.com/pdf/solution-profiles/ssl-offload-performance-sp.pdf`, June 2011.

[18] Jian Song and Yanchun Zhang. Architecture of a web accelerator for wireless networks. In *Proceedings of the thirtieth Australasian conference on Computer science - Volume 62*, ACSC '07, pages 125–129, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[19] Junli Yuan, Hung Chi, and Qibin Sun. A more precise model for web retrieval. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, WWW '05, pages 926–927, New York, NY, USA, 2005. ACM.

[20] Hao Che, Ye Tung, and Zhijun Wang. Hierarchical web caching systems: modeling, design and experimental results. *IEEE J.Sel. A. Commun.*, 20(7):1305–1314, September 2006.

[21] Zhimei Jiang and Leonard Kleinrock. Web prefetching in a mobile environment. *IEEE Personal Communications*, 5:25–34, 1998.

[22] Avinoam Eden, Brian Joh, and Trevor Mudge. Web latency reduction via client-side prefetching. In *Proceedings of the 2000 IEEE international Symposium on Performance Analysis of Systems and Software*, pages 193–200. IEEE, 2000.

[23] Xin Chen and Xiaodong Zhang. A popularity-based prediction model for web prefetching. *Computer*, 36(3):63–70, March 2003.

[24] S. Jae Yang, Jason Nieh, Shilpa Krishnappa, Aparna Mohla, and Mahdi Sajjadpour. Web browsing performance of wireless thin-client computing. In *In Proceedings of the Twelfth International World Wide Web Conference (WWW 2003*, 2003.

[25] Ramesh R. Sarukkai. Link prediction and path analysis using markov chains. In *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, pages 377–386, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co.

[26] Venkata Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve world wide web latency. *Computer Communication Review*, 26:22–36, 1996.

[27] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. A data mining algorithm for generalized web prefetching. *Knowledge and Data Engineering, IEEE Transactions on*, 15(5):1155 – 1169, sept.-oct. 2003.

[28] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. An argument for increasing

tcp's initial congestion window. *SIGCOMM Comput. Commun. Rev.*, 40:26–33, June 2010.

[29] Feng Qian, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, Oliver Spatscheck, and Walter Willinger. Tcp revisited: A fresh look at tcp in the wild. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 76–89, New York, NY, USA, 2009. ACM.

[30] Yong-Jin Lee and M. Atiquzzaman. Http transfer latency over sctp and tcp in slow start phase. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 196 –198, June 2007.

[31] John Heidemann, Katia Obraczka, and Joe Touch. Modeling the performance of http over several transport protocols. *IEEE/ACM Trans. Netw.*, 5:616–630, October 1997.

[32] Abhinav Pathak, Y. Angela Wang, Cheng Huang, Albert Greenberg, Y. Charlie Hu, Randy Kern, Jin Li, and Keith W. Ross. Measuring and evaluating tcp splitting for cloud services. In *Proceedings of the 11th international conference on Passive and active measurement*, PAM'10, pages 41–50, Berlin, Heidelberg, 2010. Springer-Verlag.

[33] Firefox web browser. `http://www.mozilla.org`, 2012.

[34] Internet explorer web browser. `http://windows.microsoft.com/en-US/internet-explorer/products/ie/home`, 2012.

[35] Google chrome web browser. `https://www.google.com/chrome`, 2012.

[36] Safari web browser. `http://www.apple.com/safari/`, 2012.

[37] Thomas Powell. 15 secretsof next-gen browsers. *NetworkWorld*, pages 402 – 416, April 2009.

[38] Mikhail Mikhailov and Craig E. Wills. Embedded objects in web pages. *Technical Report WPI-CS-TR-00-05, Computer Science Department, Worcester Polytechnic Institute*, March 2000.

[39] Sina Khanifar. San francisco cell network report. `http://opensignalmaps.com/reports/sf-april-2011.pdf`, April 2011.

[40] Steve Lohr. For impatient web users, an eye blink is just too long to wait. *New Yor Times*, February 2012.

[41] Kevin Freitas. Linkchecker. `http://www.kevinfreitas.net/extensions/linkchecker`, October 2006.

[42] Antoine Mechelynck. Xul (xml user interface language). `https://developer.mozilla.org/en/XUL`, June 2011.

[43] Mark Giffin. Javascript. `https://developer.mozilla.org/en/JavaScript`, May 2012.

[44] Bert Bos. Css (cascading style sheets). `http://www.w3.org/Style/CSS`, May 2012.

[45] Jason Yeo. Xpcom (cross platform component object model). `https://developer.mozilla.org/en/XPCOM`, November 2011.

[46] Eric Shepherd. Web workers. `https://developer.mozilla.org/En/DOM/Worker`, November 2011.

[47] Gecko:home page. `https://wiki.mozilla.org/Gecko:Home_Page`, 2012.

[48] Hung Le. Lori (life-of-request info). `http://www.le.com/~hle/lori`, May 2008.

[49] Vmware. `http://www.vmware.com`, 2012.

[50] Http archive. `http://httparchive.org`, 2012.

[51] Alexa top 500 global sites. `http://www.alexa.com/topsites`, 2011.

[52] Maxmind - geoip. `http://www.maxmind.com/app/ip-location`, 2012.