Binoy Chemmagate

An Experimental Study of Web Transport Protocols in Cellular Networks

Faculty of Electronics, Communications and Automation

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 14.12.2011

Thesis Supervisor:

Dr. Lars Eggert, Aalto University

Thesis Instructor:

M.Sc. (Tech.) Markus Isomaki, Nokia

Aalto University
School of
Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Binoy Chemmagate

Title: An Experimental Study of Web Transport Protocols in Cellular Networks

Date: 14.12.2011 Language: English Number of pages:10+66

Faculty of Electronics, Communications and Automation

Department of Communications and Networking

Professorship: Networking Technology Code: S-38

Supervisor: Dr. Lars Eggert, Aalto University

Instructor: M.Sc. (Tech.) Markus Isomaki, Nokia

HTTP and TCP have been the backbone of web transport for decades. There have been numerous enhancements and modifications to both of these protocols. HTTP and TCP were developed for traditional packet networks existing since 1990's. Today, however, wired network parameters such as bandwidth and delay have significantly improved all over the world. However, cellular data networks (GPRS, HSPA) still experience bandwidth and delay issues, which affect the performance of these protocols. HTTP and TCP protocols can be optimized for today's network conditions and end-user requirements, such as accelerated page loading, low latency and better network utilization.

Through the course of this work, we measure the improvements in using the SPDY protocol in comparison to HTTP. We measure the impact of header compression, number of parallel TCP connection per domain, and multiplexing of streams. From the TCP perspective, we analyze the impact of higher initial congestion windows. Some of the interesting findings are discussed, comparing various initial congestion window values. All of these experiments are conducted over live GPRS, HSPA and LTE networks. We study the challenges of moving from HTTP to alternative protocols. We also discuss the ways to improve the mobile web browsing by introducing and refining the existing schemes such as DNS pre-fetching, radio transition delays, smart use of IP versions, reduction of TLS negotiation delays, and intelligent allocation of TCP connections in HTTP.

Our studies reveal that low bandwidth networks such as GPRS benefits from header compression, whereas the HSPA and LTE networks benefit from multiplexing as it saves the time for establishing new TCP connections. The advantage of higher TCP initial congestion window is seen only in networks with high bandwidth and high latency.

Keywords: HTTP, SPDY, Mobile, Web, Browsers, GPRS, HSPA, LTE, TCP, Initial Congestion Window, Header Compression, Multiplexing

Preface

This thesis work is undertaken for fulfillment of the requirements of the Master's Degree Program in Communications Engineering at Aalto University School of Electrical Engineering, Finland. The task was performed at the research premises of Nokia, Espoo, Finland.

The thesis is organized in such a manner as to provide an incremental approach to the problem studied, so that the reader can gain an in-depth understanding of the research topic. The literature is structured as follows:

- Section 1 discusses the motivation for this topic for research, defining the objectives of the study undertaken, and what are the current challenges in mobile web browsing.
- Section 2 describes the background study. This section discusses the various layers of mobile web browsing and explains how the system works today.
- Section 3 discusses the shortcomings in current protocols, such as HTTP and TCP. The section also discusses improvements that can be made to this existing protocols.
- Section 4 describes the test-setup, network architecture, and the experimental methodology. The detailed statistics about the websites tested are presented in this section as well.
- Section 5 presents the performance results of SPDY in comparison to HTTP; also measures the effect of multiplexing, efficient use of TCP connections, and increased TCP initial congestion windows.
- Section 6 describes the prior work in the area of mobile web browsing and discusses the future work.
- Section 7 presents the facts and lessons from the experiments conducted.

Acknowledgments

Firstly, I would like to thank my instructor Markus Isomaki with all my heart for his patience, continuous support and guidance during the course of this thesis work. I would like to thank my supervisor Dr.Lars Eggert for his supervision and for being a such great source of encouragement, ideas and suggestions for the thesis. I would also like to express my sincere gratitude towards Simo Veikkolainen, Jari Vainikka and Timo Ali-Vehmas for considering me eligible for this wonderful opportunity.

At University of Helsinki, I am deeply indebted to Aki Nyrhinen for supporting me, helping with the programming techniques and teaching me a lot of new things about the Linux kernel. I would like to thank Markku Kojo for his valuable suggestions and support on this research topic.

It was a great pleasure to have friends like Koushik Reddy and Prashanth Pattabiraman continuously motivating me throughout the course of this work. Furthermore, I appreciate my friend Athul Prasad and my father Kunhi Chemmagate for their efforts in revising the final draft. I would also like to thank Anita Bisi, Jenni Tulensalo and Sanna Patana for extending me their wholehearted support throughout the course of my studies in Finland.

Finally, but with greatest regard I would like to express my gratitude towards my grandparents, parents and my sisters for their love and support at every step of my life. During this thesis work, they had continuously motivated me and showered me with their invaluable warmth and encouragement.

Espoo, 14.12.2011

Binoy Chemmagate.

Contents

List of Acronyms

3GPP	Third Generation Partnership Project
BDP	Bandwidth Delay Product
CDN	Content Delivery Network
CSS	Cascading Style Sheets
DDoS	Distributed Denial-of-Service
DL	Downlink
DNS	Domain Name System
DOM	Document Object Model
DRX	Discontinuous Reception
DSL	Digital Subscriber Line
EDGE	Enhanced Data rates for GSM Evolution
FTP	File Transfer Protocol
GPRS	General Packet Radio Service
GW	Gateway
HSPA	High Speed Packet Access
HTML	UMTS Terrestrial Radio Access Network
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISP	Internet Service Provider
LTE	Long Term Evolution
LTE-A	Long Term Evolution - Advanced
MAC	Medium Access Control
MSS	Maximum Segment Size
NAT	Network Address Translation
NRC	Nokia Research Center
P2P	Peer-to-Peer
PDU	Protocol Data Unit
PHY	Physical Layer
QoS	Quality of Service
RFC	Request For Comments
RLC	Radio Link Control
RNC	Radio Network Controller

RTO	Retransmission Timeout
RTT	Round Trip Time
SCTP	Stream Control Transmission Protocol
SPDY	Google Proposed SPDY Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
SST	Structured Stream Transport
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UE	User Equipment
UL	Uplink
UTRAN	UMTS Terrestrial Radio Access Network
WAP	Wireless Application Protocol
WCDMA	Wideband Code Division Multiple Access
WLAN	Wireless Local Area Network

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The past decade has witnessed an explosive growth in Internet access from smart phones [13]. Affordable flat rates and the introduction of smart phones with support for higher data rates have enhanced mobile web browsing. The network evolution started with General Packet Radio Service (GPRS) and has finally reached Long Term Evolution (LTE). The data rates have gone from hundreds of Kbps to tens of Mbps. Although the mobile phones supporting higher data rates of up to 100 Mbps are not available in the market, there is a considerable amount of people who are using mobile phones as their primary web browsing device. Mobile web browsing in GPRS was slow for many years, whereas with EDGE, the mobile web browsing experience was enhanced. The introduction of High Speed Packet Access (HSPA), with better channel coding schemes and bandwidth, made mobile web browsing a pleasant experience. Now, with the release of LTE, mobile web browsing speeds can even compete with wired networks such as DSL or Cable connections. Though extensive scrolling and readability is still an issue in mobile phone browsing, the higher bandwidth has managed to minimize the page load times. Applications such as e-mail, VoIP, and e-commerce websites have found their way from PC to the mobile, with technological changes in the cellular data networks.

Apart from data network improvements, there were changes in mobile phone hardware, operating systems, browsers, underlying protocols, and services. If the first few smart phones had only small sized screens, now the screen sizes approaches 3 inches or more. If we take mobile browsers, the choice extends from the Operating System (OS) specific browsers to migrated browsers from Personal Computers (PC). Similar to PC based web browsers, most of the mobile web browsers are based on Webkit [38] or other dominant layout engines such as Presto [10], Gecko [39], and Trident [36]. Even though the layout engine differs, the browsing experience is almost the same in most of the mobile web browsers. Some mobile browsers are pre-installed with the OS and some are installed by the user. Changes to the underlying protocols also have been incremental and has helped mobile web browsing considerably. The following sections will describe the underlying protocols, which is the core area of interest of this thesis.

The underlying protocols we consider in mobile web browsing are Domain Name System (DNS), Hypertext Transfer Protocol (HTTP), and Transmission Control Protocol (TCP). DNS is responsible for resolving domain names into network IP addresses. The DNS accommodates and resolves both IPv4 and IPv6 addresses. There have been many changes to DNS from its original form. DNS is vulnerable to many issues [50], the slow update of DNS records leading to stale caches, susceptibility to Distributed Denial of Dervice (DDoS) attacks and most importantly, the resolving process takes longer time whether it is iterative or recursive. The main concern with mobile web browsing regarding DNS is the resolution time. There have been some attempts to solve this issue by offering alternate DNS services, enhancing DNS services and DNS prefetching. DNS-prefetching offers faster browsing experience compared to the normal systems. The thesis explores the DNS-prefetching feature more during the course of this work.

HTTP is a prominent request-response protocol in web browsing which is widely used. There have been other protocols for accessing information from the Internet such as File Transfer Protocol (FTP), Secure Shell (SSH), Telnet, etc. HTTP has been the only protocol in use when it comes to web browsing and information retrieval. HTTP has evolved from HTTP/0.9 to HTTP/1.1 with many improvements and changes from its basic structure. The flexibility and importance of HTTP is such that, many network administrators have given a firewall exception on port 80 for HTTP traffic. Many applications use the HTTP protocol, as they need a clear communication port to pass through the strict firewalls, router access lists, and Network Address Translation (NAT). Although the popularity and acceptance of HTTP is higher compared to any other application layer protocols, it has some shortcomings, which can be rectified.

The reliability, congestion control and functional stability makes TCP protocol the most prominent transport layer protocol. The reliability of TCP makes it a perfect candidate for transporting the web browsing traffic. TCP has undergone various changes from its basic structure in terms of congestion control algorithms. Although there have been many advantages with TCP there are some disadvantages as well. Most of the web transactions are short lived. TCP has a property of delivering the maximum data rate for long-lived transactions as it uses a congestion control algorithm. In this case, the beneficiaries are the users who download big files using TCP, than the users who do web browsing which are short-lived transactions. The network conditions and parameters have changed over the years. Recently, there has been many proposals [16] [28] and discussion [26] whether to increase the initial congestion window to a higher value or to adopt an adaptive scheme of increasing the TCP's initial congestion value. Higher initial congestion window benefits from a low start-up delay (slow start in TCP) and higher initial data transfer rate. The advantages and disadvantages of having higher initial congestion window are discussed in chapter 3.

Most of the Internet data comes under the category of web browsing if we do not consider the Peer-to-Peer (P2P) shared music and videos. We are focusing on the size of web pages which averages to 320KB on the wire/medium as per the Google web metrics [51] and the transformation undergone by web pages. In the beginning, web pages were pure HyperText Markup Language (HTML) pages with no java scripts or Cascading Style Sheets (CSS). However, as soon as the JavaScript, CSS and other scripting languages came into use, the web content structure started to evolve. Although the basic structure is still built on HTML, the content includes CSS, java scripts, images, videos, audios, flash, etc. The web page layout, scripts, and many other resources constitute to the overall web content. To optimize the mobile web browsing, some websites offer a mobile version of the website. There is a .mobi top-level domain pointing to the mobile websites. One more technology that was dominant in the late 90's was Wireless Application Protocol (WAP), which is an open standard for mobile web browsing.

Apart from changing the content nature and structure, another significant characteristic of web content is the domain sharding [52]. For a particular website, its content does not necessarily reside in a single domain. The content can be distributed across multiple domains and are called sub-domains. These sub-domains are used for load balancing and performance improvement. Load on a server increases when a browser requests all the resources from a single server. In order to avoid this, the resources are distributed across multiple sub-domains. Primary domain or server gives the links to those distributed resources to the browser.

A typical browser makes more than one TCP connection per domain. The maximum number of TCP connections per domain depends on the browser used. When the number of sub-domains increase the number of TCP connections for a particular website also increases, this results in better performance as the browser is able to request multiple objects through multiple TCP connections. Number of TCP connections and sub-domain relation will be discussed further in this thesis. There has been research [53] [55] and implementation ongoing in this area of mobile web browsing from various scientific societies, such as Internet Engineering Task Force (IETF). Most of them are based on the simulated environment. The results obtained from that research cannot be directly applicable to real world scenarios. This was one of the main motivations to conduct an experimental study on mobile web browsing in real world scenario.

Considering all the facts presented above, it was a essential to know whether changes made to traditional protocols like HTTP and TCP, in practice, bring any positive changes to the user experience. The users measure the performance of mobile web browsing in terms of page load time. Page load time can be subjective as there is an instance when the browser renders particular objects and the user is able to navigate through the page even before the browser renders all the other objects. Subjective measurements depend on browsers and website structure.

1.2 Objectives

The primary objective of this thesis work was to conduct a comprehensive study on the improvements in mobile web browsing in real-time cellular networks. The areas of interest being, the delay associated with every step in mobile web browsing, starting with radio access and ending with browser rendering. First part of the thesis introduces mobile web browsing. Second part concentrates mainly on the experiments conducted and the discussions related to the results obtained.

In the chapter 3, We discuss HTTP and TCP protocol shortcomings and the Google proposed protocol SPDY [35]. Numerous tests have been conducted comparing the HTTP and SPDY protocols. We analyze the performance of SPDY on the cellular networks and discusse the benefits of SPDY features. Another parameter of the experiment is TCP's higher initial window. The focus was to measure the impact of increased TCP initial congestion window in GPRS, HSPA, and LTE networks.

Other areas of improvement mainly deal with radio channel transition, which cover the UMTS, GSM, and LTE state machines, IPv4 and IPv6 preferences, introduction to DNS services which mainly deals with DNS prefetching, and delay in TLS negotiation is explored as security is an important part in mobile web browsing. There are no measurements conducted in the above-mentioned areas but we give a short introduction to each of the above mentioned areas.

2 Background

This section describes the critical elements in mobile web browsing. It covers the mobile web browsing steps in sequence. Every step is explained in detail with diagrams and background information.

2.1 Introduction to Mobile Web Browsing

Once the mobile phone and browser are in place, the next essential thing is an Internet connection. The introduction of HSPA and LTE changed the face of mobile web browsing as the bandwidth was no longer a limitation. HSPA offers up to 14 Mbps (theoretical values) downlink in some countries and LTE offers up to 300 Mbps downlink (theoretical values). Depending on the market and availability, GPRS/EDGE are still used in many of the developing countries.

Access Technologies and Speed		
Technology/Standard	Downlink (Theoretical)	Uplink (Theoretical)
GPRS (Release 97)	40 Kbps - 171 Kbps	14 Kbps
EDGE [1]	384 Kbps	135 Kbps
HSPA [2]	14 Mbps	5.8 Mbps
HSPA+	42 Mbps	11 Mbps
LTE	300 Mbps	75 Mbps

Table 1: Access Technologies and Speed

2.2 Web Browsing in Cellular Networks

A brief overview of how the web browsing actually works is presented in following sub-sections. Access networks such as GPRS, EDGE, HSPA, and LTE have different state machines. The state machines we discuss here are mainly for HSPA networks. The mobile web browsing involves several entities like radio channel allocation, DNS queries, TCP connection establishment, SSL/TLS negotiation, HTTP request and responses, and browser rendering. There are many elements and parameters that can be optimized to gain maximum performance in a mobile web browsing. The steps below are based on the assumption that the browser is open and awaiting input from user.

Step 0 User indicates the desired the website to the mobile browser and hits the Go/Enter.

Step 1 DNS resolver checks the DNS cache for relevant information, if the DNS cache is missed then a DNS query is sent to the Internet to resolve the domain name to network IP address.

Step 2 If the radio is not active then the DNS query from the DNS resolver invokes the radio in the mobile device, prompting the user to connect to the radio bearer.

Step 3 Once the mobile device is connected to the radio bearer, it is assigned a public or private IP address by the network operator. This public or private IP address can be IPv4 or IPv6 depending on the operator and availability.

Step 4 Radio state transition starts, initially the radio is in IDLE state, as soon as the data transfer begins state promotions happen and radio gets promoted to FACH or DCH states.

Step 5 DNS query is sent to the network and DNS response arrives with an IPv4 and/or IPv6 address(es). Such data transfer causes the radio state to get promoted from IDLE to FACH or DCH.

Step 6 As soon as the IP address(es) of website are available to the browser, it checks for existing TCP connections and tries to re-use the TCP connection if available. In any other case, a TCP connection is established between the mobile device browser and web server.

Step 7 Depending on the security policy adopted by the website (http or https), the connection can be secure or non-secure. The browser initiates a handshake with web server for a certificate exchange to verify the authenticity of the website.

Step 8 HTTP requests and responses begin. Once the TCP connection is established and SSL/TLS negotiation is over, the next step is to request resources from the web server. Web server responds with corresponding resources in sync with web browser requests.

Step 9 Browser renders the objects retrieved by HTTP protocol on the mobile device screen. When browser receives the resources or objects from the web server, it renders the object using the layout engine associated with it.

Step 10 The DNS resolution and Steps 6, 7, 8, and 9 are repeated for every object in the website until the website is completely retrieved from the server.

From the Figure 1, we can see that, user requests for a particular website www.example.com by typing the address in browsers address bar. DNS resolver program in mobile device invokes the radio of mobile device prompting the user to connect to the radio bearer. The UDP packet from the DNS resolver causes the radio channel transitions from IDLE to FACH. Once the radio channel transitions are completed, the DNS query gets forwarded to resolve the www.example.com. When the desired IP address(es) is obtained using DNS resolution, mobile device makes a TCP connection to the particular server. HTTP GET request is sent for the index.html file. The HTTP response causes the radio channel transition from

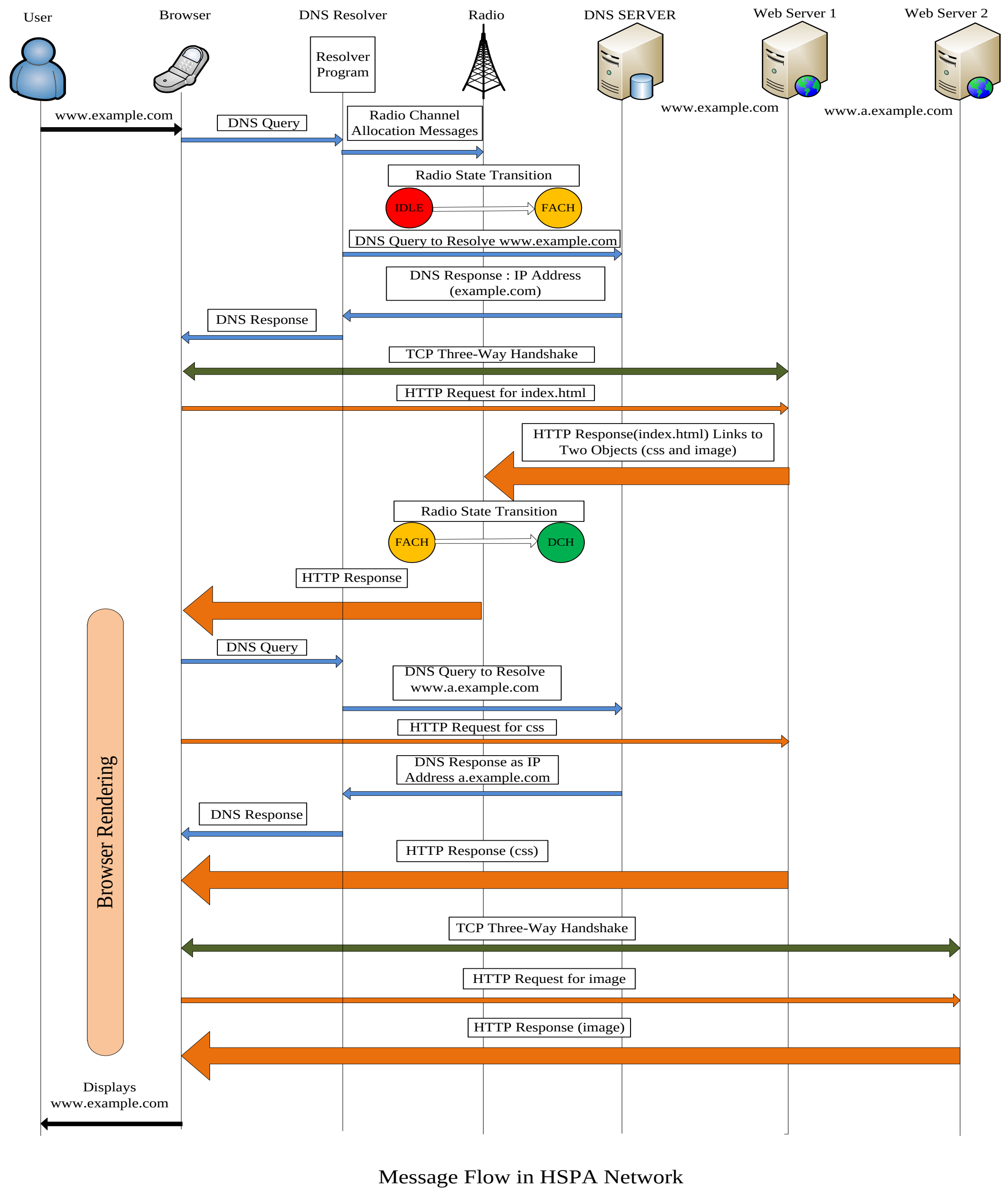

Figure 1: Message Flow in Mobile Web Browsing

FACH to DCH. The mobile browser learns about the two objects (image and CSS) present in the index.html. Since the CSS is in the same domain as index.html, another request is issued to retrieve the CSS. At the same time, a DNS query is sent to resolve the sub-domain address www.a.example.com, where the image resides. When the desired IP address(es) is obtained from the DNS resolution, a second TCP connection is established between the sub-domain and mobile device.

Mobile device requests for the image and retrieves the image. The user can see the complete website www.example.com once all the objects are retrieved.

2.2.1 HTTP Requests and Responses

HTTP protocol is based on request-response model. Client initiates a request and server responds. When the user requests for a particular web page, server responds with the page and resources associated with the page. The browser might have to send multiple requests depending on the number of objects present in the website. For example, if an index page contains one image, text and some CSS files, then the browser will first request for the index page with text and it will make subsequent request for image and css. Once the objects are received, the browser will render the objects as intended by the website structure.

The delay in this model is mainly the resource retrieval time or network latency. In case of resource retrieval, time the main issue can be a bad cache control. Cache control is a rule based approach hence it should be efficient. A good cache control method can reduce the load on web server largely. Comprehensive compression methods like Gzip can significantly reduce the payload of HTTP and thereby reduce the effect of high network latency. However, the compression method should be agreed between client and server beforehand.

2.2.2 Delays in Radio Access

When the mobile device establishes a connection with the radio bearer, it does not imply that user can immediately start browsing. The UMTS radio bearer has three states, they are IDLE, FACH, and DCH as seen on Figure 2. The three states are essential for effective resource utilization and energy efficiency. IDLE is the state when mobile device is connected to radio bearer but no data channel is allocated to the device. In IDLE state, mobile device cannot transfer any user data. In FACH state, the mobile device can transmit user data in low speed channels and there is no dedicated channel available for the mobile device. In DCH state, the mobile device is allocated with a dedicated channel for both downlink and uplink.

When a user starts to browse for the first time, ideally the mobile device is in IDLE state. When mobile device sends user data to radio network, it moves from IDLE to FACH and later to DCH. The time to move from one state to a higher or lower state depends on the network configuration. Studies show that IDLE to DCH can take up to 2 seconds [47] whereas moving from DCH to IDLE can take up to 10-17 seconds. The Figure 2 explains the channel transitions in HSPA radio network. Apart from the IDLE channel, some network vendors offer intermediate states called CELL_PCH and URA_PCH. These intermediate states help to overcome the issues with fast dormancy [31].

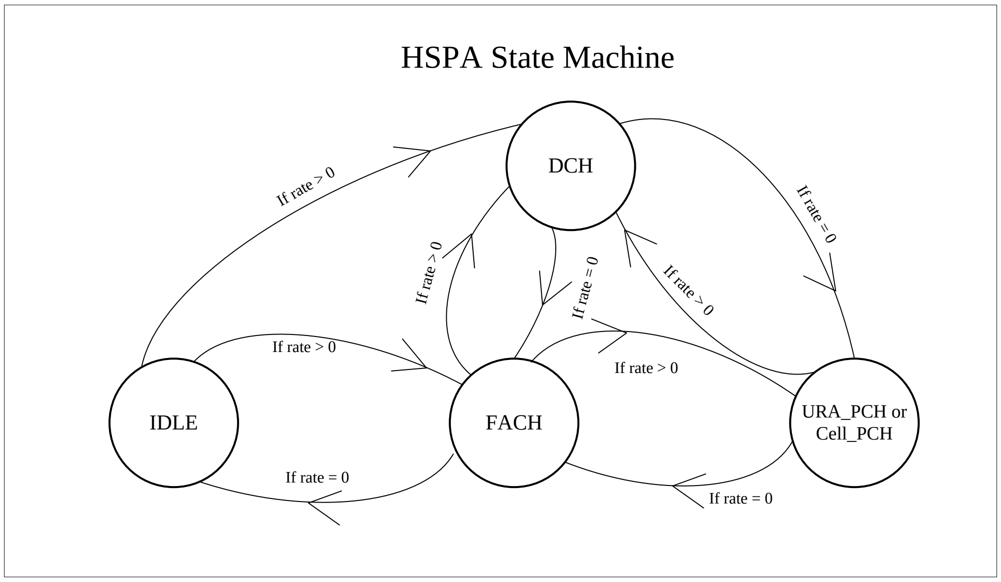

Figure 2: Channel Transition from IDLE to FACH and DCH [44]

We discussed about different state machines in HSPA. The state transition plays a vital role in managing the data transfer and energy efficiency. If the mobile device requires faster data transfer, it has to be in DCH state even though DCH state consumes more power than other states. Two types of state transitions are available in HSPA, one is state promotions where radio state moves from lower states like IDLE or FACH to DCH and second is state demotions when radio state moves from DCH to FACH or IDLE. When mobile device is sending continuous packets to network, it should be in DCH state. When the mobile device does not have any packet to send, it moves to FACH or IDLE. The network decides state promotion and state demotion time out periods. In the case of state demotion, if the timeout is a large value, power consumption of mobile device is high. Although mobile device saves some power when timeout is low, if the mobile device has some data to be sent immediately to network, it has to wait until the mobile device moves to higher state. When we consider state promotions, large timeout values can be a hindrance for sending user data immediately to network. Continuous state promotions and demotions can be an overhead for the radio network.

The GPRS network also maintains three states corresponding to UMTS network as IDLE, READY, and STANDBY. The Figure 3 describes the state machines in GPRS. In IDLE state, there is no data transfer. When there is data to send to GPRS network, an attach procedure is done and device moves to READY state. When there is no data to send or when the READY timer expires, radio bearer switches to STANDBY state. The STANDBY state goes back to READY state when there are Protocol Data Unit (PDU) transmissions. The GPRS detach procedure is performed when mobile device is disconnected from GPRS network.

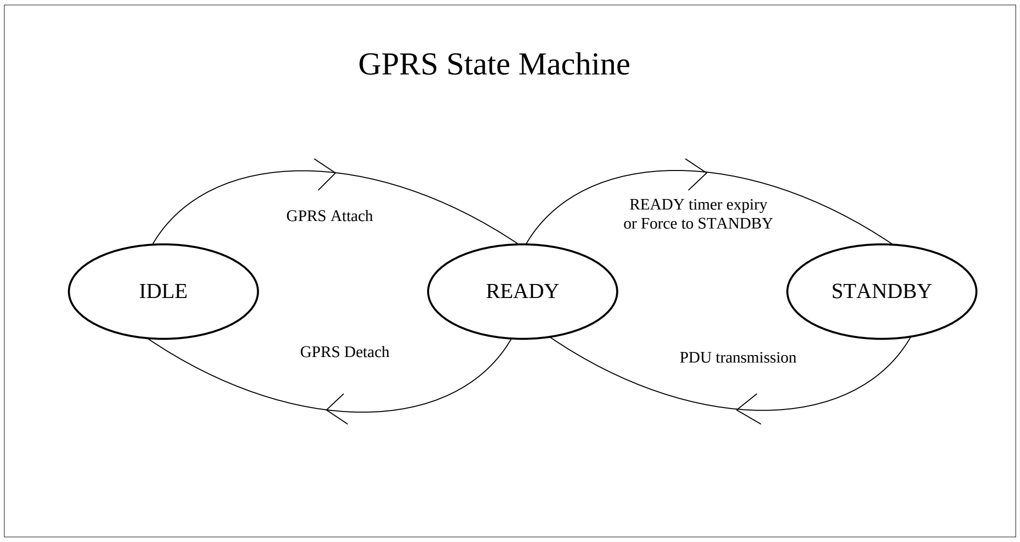

Figure 3: Channel Transition in GPRS [32]

The LTE network maintains two states RRC_IDLE and RRC_CONNECTED. The Figure 4 describes the state machines in LTE. In RRC_IDLE device sleeps for most of the time and occasional uplink transmission is possible through random access. Moving from RRC_IDLE to RRC_CONNECTED takes time compared to leaving Discontinuous Reception (DRX). In RRC_CONNECTED, there are two substates called OUT_OF_SYNC and IN_SYNC. In the former case, only downlink transmission is possible and no uplink transmission is possible whereas in latter case, both downlink and uplink transmission is possible.

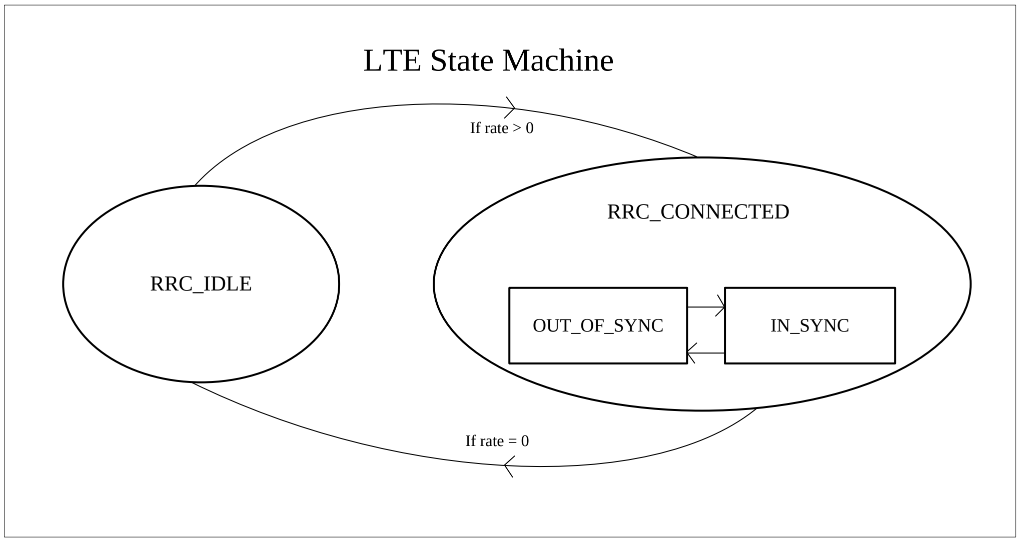

Figure 4: Channel Transition in LTE [17]

There has been few proposals [62] and studies [46] [11] in the field of radio resource allocation delays. One study conducted on U.S. based operators shows that state demotion timeout can be up to several seconds depending on the carrier. One of the interesting facts from this study is that a state promotion in one carrier takes up to 2 seconds. If the mobile device is able to move to DCH beforehand it saves 2 seconds and user experiences a rapid page load time. Moving to the DCH state beforehand can avoid the state promotion delays. For example, when user opens the web browser, it can force the radio state to move from IDLE to DCH. Either the mobile device can inform the RNC about the futuristic data transfer or send some dummy packets to network. Another effective way would be to use DNS prefetching for the state promotions.

The radio state transition delays can play a major role in page load time. In cases where the overall page load time is close to 4 to 5 seconds, the radio transition delay can be 50% of the total page load time. Hence, the radio delay can have a considerable impact on small size websites where the page load time is small. The radio state transitions also cause inefficient energy utilization of mobile device on old radio networks.

2.2.3 IPv4/IPv6

Depending on the mobile device capability, operator provisions, and support from the web servers or services, a mobile device can use either IPv4 or IPv6 protocol. Currently, IPv4 is the most widely used protocol. IPv6 capable mobile handsets are rare and Internet Service Providers (ISPs) offering the IPv6 address is not very common. There is no concrete evidence for quality or change in browsing experience in using IPv6 over IPv4. Some mobile device supports both IPv4 and IPv6 protocol, which is called dual stack. Using dual stack, the mobile device can use both IPv4 and IPv6 technologies to connect to destinations. It can use IPv4, IPv6 or both in parallel. When a DNS query returns IPv4 and IPv6 addresses, by rfc3484 [15] the mobile device may choose IPv6 over IPv4. The architecture of dual stack is given in Figure 5. If the network or web server does not support IPv6, the browser can fall back to IPv4 address. The time for this fall back is a good measurement study [60] in mobile phones.

2.2.4 IPv4/IPv6 Fall Back

As far the user experience is concerned there is no significant improvement in using IPv6 protocol. Sometimes, the IPv6 supported domain names are not reachable or the service is not available on IPv6. Browsers need to fall back to IPv4 as a reactive measure. IE9 and Opera 11 seems to learn [60] from the broken links and fall back to IPv4 whereas browsers like Chrome make an IPv4 connection in parallel with IPv6 and whichever connection succeeds first will be given priority. Popular browsers like Firefox and Safari face serious delays while falling back to IPv4.

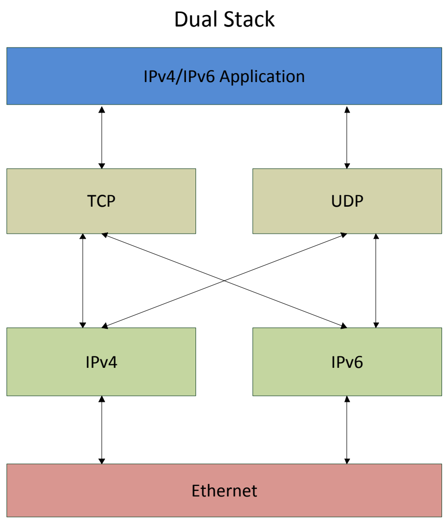

Figure 5: IPv4/IPv6 Dual Stack

The IPv6 is not supported by most of the web servers and operators. The IPv6 to IPv4 fall back is important only if the device has IPv6 capability and the device wants to access a service on top of IPv6. Most of the web browsers support IPv6 but still use IPv4 as the main IP version. One of the initiative from networking industry on this area is the Happy Eyeballs [14]. Dual stack is capable of providing IPv4 and IPv6 DNS records to the system. Happy Eyeballs proposes to establish the TCP connection using both IPv4 and IPv6. A stateful behavior can remember which protocol was successful or failure on particular network prefixes.

2.2.5 DNS Services

The DNS plays an essential part in resolving the domain names in to IP addresses. In fixed and mobile web browsing, DNS delays can be a crucial measure. When user indicates a website address to the browser, a DNS query is sent to DNS server for translating the domains name to the IP address(es). The resolving process can take from milliseconds to few seconds depending on the bandwidth, Round Trip Time (RTT), type of queries (recursive or iterative), and the level of cached DNS records. Depending on the load balancing strategies applied, single DNS name can return multiple IP addresses. The browser can use any of these IP addresses for connecting to the web server. DNS queries can be iterative or recursive. Iterative queries are made from a DNS client to DNS server, DNS server refers to another DNS server, and the referral goes on until the DNS client finds the authoritative server for the website queried. In recursive query, DNS client sends requests to DNS server, DNS server on behalf of DNS client will inquire other DNS servers, and the query goes from one DNS server to another until it finds the authoritative server for the website requested. The DNS response returns to the DNS client in the same order as request was routed. The Figure 6 is an example of iterative DNS query.

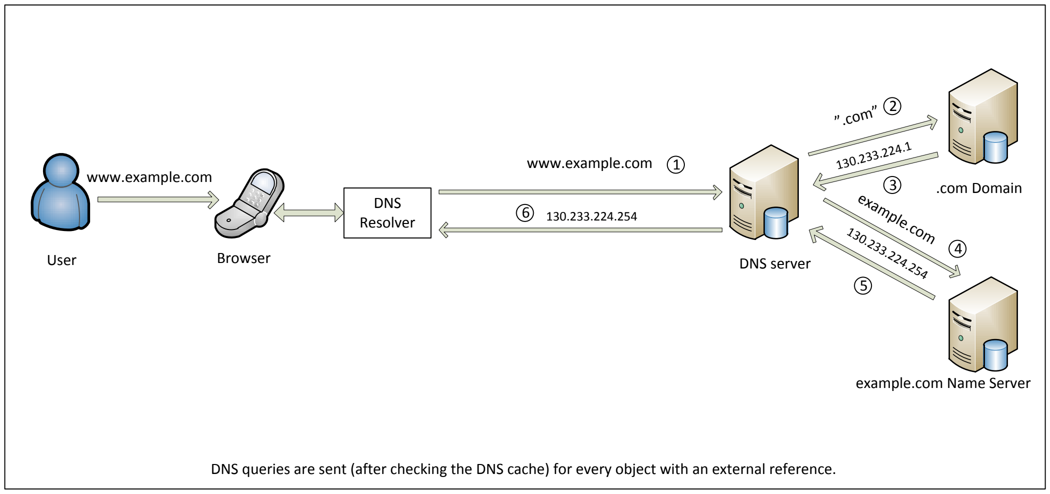

Figure 6: DNS Query and Response

Two types of delay can be considered in DNS, related to this thesis. First is the DNS propagation delay. Second is the DNS delay due to the RTT between mobile device and DNS server. The main reason for DNS propagation delay is the time required to update the cache of DNS records. When there is change from the Domain Registrars, The Root Name Servers update their DNS records. This change propagates to major DNS server and ISP DNS records. Typically, ISP's maintains a DNS cache to reduce the DNS lookup time. This cache should be up to date if the changes have to be seen by the end user. Coming back to user experience, the propagation delay can cause the ISP's to return stale DNS records. The RTT between mobile device and DNS server is another issue, if the DNS query misses the ISP's DNS cache, then the query is forwarded to the DNS servers. This process can take up to several seconds depending on the RTT between mobile device and DNS server. This can affect the user experience.

We have discussed about the DNS delay and how it can affect the user experience. DNS prefetching enables the browsers to resolve the domain names to IP address(es) even before the browser requests a particular domain name to be resolved. There are many ways to achieve the DNS prefetching. First, the manual prefetching, where the HTML page has the tag called dns-prefetch, domain names with this tag can be immediately resolved as soon as the browser parses the HTML page. Second, resolve the domain names of recently used websites during browser start-up. In this case, frequently used websites are accessible faster. Third, resolve the domain names when user types them into the address bar. The browser makes one or two suggestion based on the first few letters of the domain name and resolves them beforehand. The DNS-prefetch reduces the DNS latency, when it comes to mobile devices there are other benefits as well. When browser makes DNS queries as a part of DNS-prefecthing, the mobile device sends packets to the cellular network waking up the radio. This start-up data flow keeps the mobile device always in higher states like FACH or DCH and avoids the state promotions delays.

2.2.6 TCP Connections

Once the IP address or the list of IP addresses of the domain name is available, the next step is to establish a TCP connection to one of the servers. The browser opens a TCP connection to the web server by sending a SYN segment. The server returns a SYN-ACK segment on successful connection establishment. The mobile device can send ACK and HTTP requests (data) immediately after receiving the SYN-ACK. Depending on the number of objects and sub-domains; the browser opens multiple TCP connections. The number of parallel TCP connection per domain has grown to 6 or more (depends on browser). The number of TCP connections depends also on the browser, website structure and most importantly the bandwidth. Parallel TCP connection to a website and its sub-domains helps the browsers to request multiple objects in parallel from the website.

One of the main characteristic of TCP is its congestion control algorithms. The main congestion control mechanisms in TCP are slow start [58], congestion avoidance, fast retransmit, and fast recovery. During the slow start, TCP sender starts with an initial window of one and for every ACK it receives, the congestion window increases by one segment. When the congestion window reaches two after receiving two ACK's from the receiver, the sender can send two segments. The window grows exponentially as one, two, four, and eight until it reaches the threshold value. Once the threshold reaches, TCP moves to congestion avoidance phase where the congestion window increases by one segment for every ACK received. If any packet loss is detected, then depending on the version of TCP used reactive measures are taken.

If TCP has a high initial congestion window then slow start will begin with the high initial congestion window value. In theory, the number of segments sent in the initial burst will be high. The high initial congestion window grows faster and reaches the threshold. The head start of high initial congestion window helps TCP to reach maximum data rate in less time compared to small initial congestion window values. This practice can lead to an early page load for small sized websites and also speed up the TCP to reach the maximum throughput, if there is no congestion. If a domain has three sub-domains then the browser can make 18 parallel TCP connections. 18 parallel connections with an initial congestion window of 3 effectively boost the overall initial congestion window value to 54.

2.2.7 SSL/TLS

SSL/TLS are cryptographic protocols used in Internet for communication security. The process involves a complex handshake between client and server to verify the authenticity of website and ensures data security. The handshake provides a key, used to encrypt and decrypt the data exchanged between the client and the server. The extra security provided by SSL/TLS adds up more delay. Two main processes

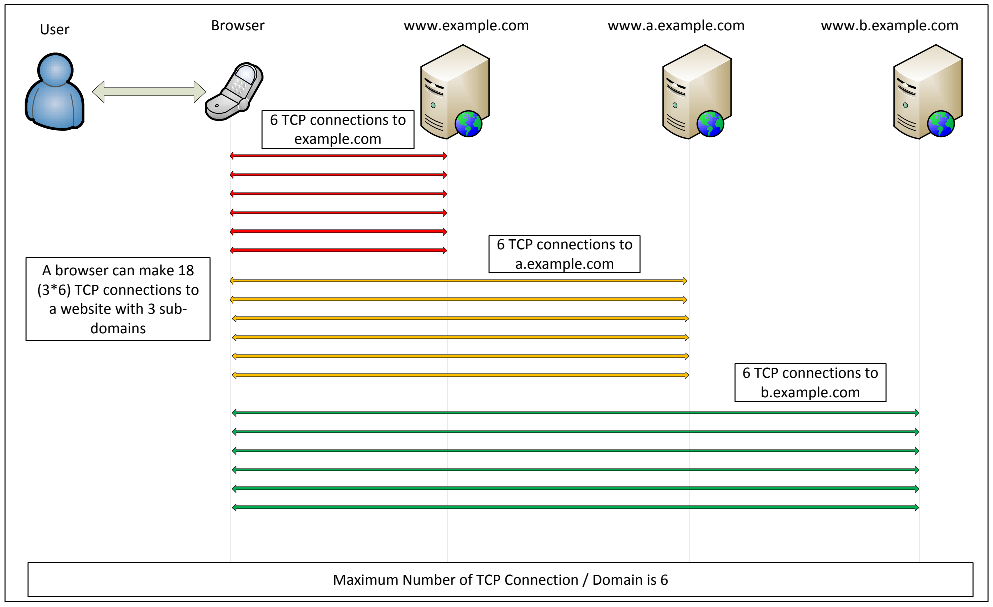
User
Browser
www.example.com
www.a.example.com
www.b.example.com
6 TCP connections to example.com
6 TCP connections to a.example.com
6 TCP connections to b.example.com
A browser can make 18 (3*6) TCP connections to a website with 3 sub-domains
Maximum Number of TCP Connection / Domain is 6

Figure 7: TCP Connections per Domain

cause the delay. First, the extra round trips required for the handshake to establish a secure connection. Second, the processing required on both ends for encrypting and decrypting the data using the key generated during the handshake. We can ignore the computing cost in latest mobile devices, as their computing power is more than the computers that existed in 1990's. Since the TLS delay is associated with security, the delay introduced is unavoidable.

We discussed about the TLS negotiation and the delay introduced in TLS negotiation. There are drafts in IETF named snap start [5], false start [6], tcpcrypt [3] for improving the delay or replacing the security layer parameters. First, is the Snap start, which tries to reduce the delay in TLS negotiation in cases where client initiates the negotiation. Client includes application data and all the other parameters server requires to complete the negotiation. Second, is the False start, generally, TLS negotiation takes 4 flights or 2 round trips. Here the aim is to send application data in third flight as the client has finished the TLS negotiation from its part. Third, is the tcpcrypt, it allows adding encryption to TCP packets. Tcpcrypt aims to modify the current security layer implementations hence need more analysis and research to find out the benefits in reducing the delay.

2.3 Summary

In this section we discussed about the message flow in mobile web browsing, explaining the step by step procedure of mobile web browsing. The section also covered the delays in different levels of mobile web browsing, explaining the HTTP web server delays, Radio delays, DNS delays, IPv4/IPv6 transitions, the strategy used in TCP connections and the SSL/TLS mechanism. All the sub-sections analyzed how the system works and parameters or mechanisms that can be improved. In the next section, we discuss some areas, which can be improved in detail. The intention of this section was to give an overall idea about the mobile web browsing.

3 A Detailed Investigation of Current Protocols

There are numerous elements in mobile web browsing which can be optimized to achieve maximum performance. We concentrate mainly on the improvements that can be made to HTTP, analyzing alternative protocols, and the effect of TCP's initial congestion window. The following sections will discuss more about the shortcomings in current protocols, and the proposed solutions.

3.1 Shortcomings in Current Protocols

Some of the shortcomings of HTTP is listed in the following sub-sections. Current HTTP implementation in browsers can fetch only one resource at a time using a single TCP connection. Even though pipelining exists in HTTP/1.1 [19], it allows clients to send multiple requests on a single connection and the server replies to those requests in the order they are received. This might trigger Head-of-Line Blocking. To be more precise, if multiple requests are sent over the same connection then the first response can block the forthcoming responses, as the queue is always a FIFO. When we look at the implementation on servers, web browsers, and proxies, it is clear that most of them do not make use of this feature of HTTP. It is relatively easy to implement pipelining in a web server but we have to make sure the network buffers are not flushed, when queuing the pipelined requests. Web browsers like Safari [37] and Chrome [24] do not support pipelining whereas Mozilla Firefox [40] has it disabled by default and Opera [42] from version 4.0 supports pipelining by default. Among the proxies, the proxies such as squid [57] do not support or have disabled the pipelining feature.

Another feature as important as pipelining is multiplexing. HTTP/1.1 has a similar feature to multiplexing in the name of chunked transfer-encoding. The response is broken down to small chunks of data when the content length is unknown to the sender. Although chunking allows the sender to split the content and send, it does not ensure the interleaved content distribution of different resources or objects. Multiplexing allows the server to maintain the persistent TCP connections and avoids the delay in establishing a new TCP connection. Chunked transfer-encoding is not supported in HTTP/1.0 hence it is important to make sure that the client and server are using the HTTP/1.1.

The limited number of TCP connections per domain is a bottleneck as the number of concurrent requests are limited on those TCP connections. TCP's slow start algorithm also plays a key role in overall performance when the number of TCP connections per domain is small. During slow start, every TCP connection starts with a small initial window and it takes time for the TCP to reach maximum throughput. When there are multiple TCP connections, multiple objects are requested through multiple TCP connections and we can minimize the effect of slow start. The browser developers adopted the idea of increasing the number of parallel TCP connections per domain. The website content is not hosted on a single domain; rather the content is distributed over multiple sub-domains. In this case, a web browser can make multiple requests over multiple TCP connections. Since the HTTP is a stateless protocol, the server treats each request as an independent query and client receives objects in a faster way. Currently, the maximum number of TCP connections per domain has grown from 2 to 6 or 8. Opera 9 [43] is a good example of this. Multiple TCP connections per domain is not a bottleneck when it comes to high bandwidth networks with low latency.

In traditional HTTP, the client makes the requests and the server responds with resources. The server is not allowed make any request or push a resource towards the client. This has both positive and negative effects. An assumption can be that a client wants only what it requests for. The resources pushed by the server might not be useful, resulting in an additional latency due to inefficient use of bandwidth. Another assumption can be that a client requires some suggestions, meta-data, and navigation instructions through the website it visits for the first time. In HTML5 [23] specification, there is provision for loading multimedia objects or meta-data beforehand using the PRELOAD tag. In the latter case, the client would prefer some server suggestions and hints. Another addition to server push would be request prioritization. In HTTP client cannot prioritize the requests or indicate the server to send the essential objects to render the website. Even though the essential objects term is quite subjective, it could be useful for mobile devices where bandwidth is limited.

Header and data compression have been a debatable topic over the years. The question remains as what would be the gain in compressing the HTTP headers. Even though the size of HTTP headers amount in few KB's compared to the actual payload, it can have a greater impact when the bandwidth is minimal. In addition, the redundant information in headers can be removed to optimize the header contents. In HTTP, data compression is achieved by content encoding and transfer encoding in HTTP/1.1. Issues concerning Page Load Time (PLT) in HTTP have remained as a point of concern for most web developers and designers over the years. The improvement solutions have always been a) increasing the number of connections per domain, b) optimizing the resource size, and c) distributing content over multiple domains.

In contrast to HTTP, TCP has evolved independently as a main transport layer protocol ensuring reliable transport. TCP has several parameters, which are optimized by different OS's to get maximum performance. Congestion control algorithms, buffer sizes, initial congestion window are few examples. One of the major concerns of the Internet community regarding TCP is the debate on increasing the initial congestion window (*initcwnd*) [9]. The initial congestion window determines the number of packets a sender can transmit or on flight at the beginning of a TCP transaction. The most common value of *initcwnd* used by Linux is 3 segments, which is almost equal to 4KB of Data. The most interesting part is that, almost all the web transports are short-lived and they do not reach the maximum window value at any point of time [25] as the transaction finishes before the TCP can reach the maximum throughput. The slow start algorithm in TCP allows the congestion window to grow exponentially as the packets are acknowledged. When a packet loss occurs, the TCP falls in to congestion avoidance phase. There are many advantages and disadvantages of having high initial congestion window, however, is not in the scope of this thesis.

3.2 HTTP and New Proposals

This section discusses the areas of improvement in HTTP, mainly focusing on header compression, request prioritization, and multiplexing.

3.2.1 Header Compression

According the experiments conducted by Google, only two third of the actual compressible material in a web page is actually compressed [4]. Even though the Google statistics are concentrating on content encoding, we are interested in the header compression, which is approximately 200 bytes to 2KB. Both request and response header size can be reduced using header compression.

Beneficiaries of header compression are the GPRS or WCDMA modems with low up-link capacities, where the client requests face serialization latency. In addition, the redundancy in the subsequent request/response headers can be avoided after the initial negotiation. The overhead in this case is mainly on the server and client side. Server and client should agree on the particular header compression method and have straightforward rules on sending redundant header information. The processing load on end devices increases compared to the uncompressed method but the data on wire is reduced by sending small sized packets.

3.2.2 Request Prioritization

There could be instances when user is waiting for a particular resource and the browser renders that particular object at the end. For example, when we try to login to Gmail or Facebook, there are instances when the login box appears at the end and objects like images, logos, advertisement, and other irrelevant text appears first. From the end-user point of view, the user is more interested in the login box than the other objects on the website. It would be interesting if the client can set a high priority for the login box and server can push the login box first. In case the objects are present in different sub-domains, the primary server cannot interfere the transfer or push the objects. The client can request important objects using the meta-data information.

The request prioritization is a method by which client can prioritize the requests. The client can obtain critical resources by placing a priority on each request. This can be possible with pre-loading the meta-data of the website. This way client can request the resources important for rendering. The request prioritization can also be useful when user wants to navigate through the website even before it is fully loaded. This ability to navigate or use the resources before the page is fully loaded is a subjective measurement of page load time. Generally, the browsers create DOM (Document Object Model) tree before creating a render tree. A render tree is used for layout and painting the objects later. In some cases, browsers tend to show the objects on the screen as soon as the objects are received from the server before creating the render tree. The request prioritization can be used to speedup this process. In such cases, the client can request for the most essential resources for the user to browse the website, ignoring the advertisements or other irrelevant resources.

3.2.3 Multiplexing

The HTTP has the provision for pipelining but multiplexing or interleaved transactions is not implemented in HTTP. Assuming multiplexing were available in HTTP, it could send multiple objects interleaved in a single TCP connection. When client needs an object from the server, it initiates a TCP connection and requests the object from the server. If a TCP connection is already open, HTTP can make use of its persistent connection feature and request the object from server. If we consider the case, where client has already established a connection and it requires an additional object from server. The client can either reuse the existing TCP connection or open a new connection. Two possibilities are present here 1) client is in the middle of receiving an object requested previously or 2) TCP connection is idle and no data transfer is active. If we use multiplexing in the first case then the client can request the additional object using the same active TCP connection. The client saves the additional delay in establishing a new TCP connection by making use of the existing active TCP connection. When the TCP connection is idle and no user data is in flight, there is no need for multiplexing, as there is only one object to be transferred. Multiplexing also solves the Head-of-Line blocking in HTTP pipelining. Using multiplexing we can send the available objects immediately to the network without waiting for the blocking response.

The Figure 8 describes how the pipelining works, we can consider three requests (request-1, request-2 and request-3) and 3 responses (response-1, response-2 and response-3). The requests are served in the order they are received. The response-2 and response-3 has to wait if the response-1 is not available immediately.

From the Figure 9, we can see that in case of multiplexing, the response is split into smaller chunks and sent to network immediately. It can also be noted that response-1 chunks are preceded by response-2 chunks and response-3 chunks. If response-1 is not available immediately then the pending responses are prioritized instead of waiting for the response-1.

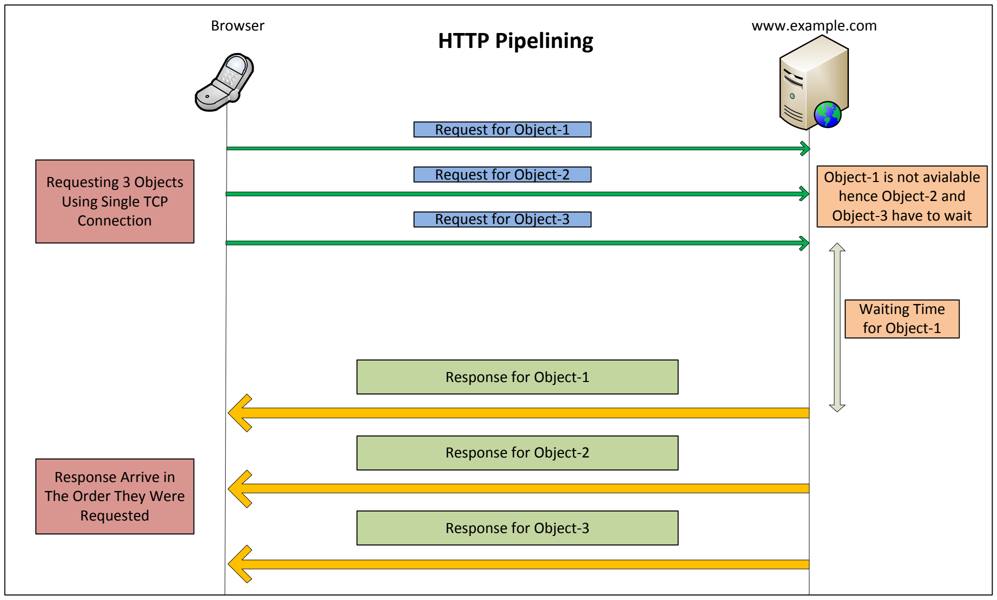
Browser
HTTP Pipelining
www.example.com
Request for Object-1
Request for Object-2
Request for Object-3
Requesting 3 Objects Using Single TCP Connection
Object-1 is not avialable hence Object-2 and Object-3 have to wait
Waiting Time for Object-1
Response for Object-1
Response for Object-2
Response for Object-3
Response Arrive in The Order They Were Requested

Figure 8: Pipelining

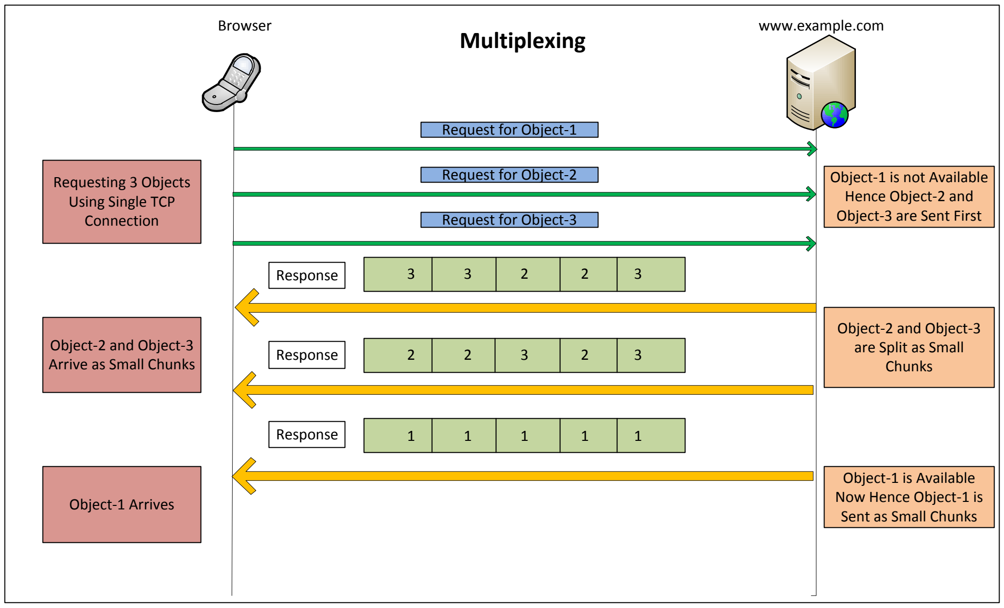
Browser
Multiplexing
www.example.com
Request for Object-1
Request for Object-2
Request for Object-3
Requesting 3 Objects Using Single TCP Connection
Object-1 is not Available Hence Object-2 and Object-3 are Sent First
Response
3 3 2 2 3
Response
2 2 3 2 3
Object-2 and Object-3 Arrive as Small Chunks
Object-2 and Object-3 are Split as Small Chunks
Response
1 1 1 1 1
Object-1 Arrives
Object-1 is Available Now Hence Object-1 is Sent as Small Chunks

Figure 9: Multiplexing

It has been a challenging question whether multiple TCP connections performs better than multiplexing. The browsers open multiple TCP connections per subdomain. Some TCP connections are active only for short period. Therefore, it would be optimal to reuse or multiplex the existing TCP connections. This way we can save the time for the new TCP connection establishment. Opening multiple TCP connection is always beneficial for the browsers when bandwidth is not a limiting factor. Multiplexing helps in low bandwidth scenarios such as GPRS and dial-up connections.

3.3 SPDY, An Alternative Protocol

SPDY is a Google proposed application layer protocol. SPDY mainly tries to reduce web latency by proposing some new techniques to improve the HTTP. There are many goals for SPDY but the main aim is to reduce the page load time of websites. SPDY makes substantial changes in the way HTTP behaves. In this thesis, we are exploring the features of SPDY and experimenting with SPDY. The main aim is to verify, whether the changes made by SPDY have any advantages on the web browsing over high latency cellular data networks. So far, the results published using SPDY protocol in comparison to HTTP was in a simulated environment. We are testing the SPDY protocol performance over live commercial cellular data networks.

3.3.1 SPDY Features

Multiplexed Streams: SPDY uses fewer TCP connections per sub-domain. Data streams are multiplexed over this single TCP connection for efficient use of TCP connection. The data streams are tightly packed and make use of the existing TCP connection. They are bidirectional data frames initiated by either client or server. Streams are identified by stream ID's. Client initiated streams have odd numbers and server initiated streams have even numbers. A single TCP connection can contains multiple streams.

Request prioritization: Client can set priority for the resources it needs from the server hence it can get the high priority resource faster than the low priority one's. The client might block the requests in order to prevent the congestion over a connection where bandwidth is limited.

Header compression: The client sends a lot of redundant header information. This information is exchanged every time client requests a new page, this data can be compressed and number bytes can be reduced by introducing header compression. Header compression can bring significant reduction in size of request and response headers.

Advanced features like server initiated data exchanges such as server push and server hint have been proposed in SPDY. The resources which server thinks important for the client can be pushed to client without client requesting for it. Server hint can be used to give signals to the client about an important resource, resulting in a client request and server response.

3.4 TCP and New Proposals

There are many parameters in TCP which are optimized by many OS's to achieve maximum performance. However, in this thesis, we are primarily focusing on the TCP's initial congestion window.

3.4.1 Initial Congestion Window

Today, the *initcwnd* value is around 3 segments (almost 4KB of data), which is the amount of data server can send during the initial phase of TCP transaction. Many of the web transactions are short lived, therefore, most of the web transactions do not reach the maximum TCP throughput. The value, *initcwnd* of 3 segments is small, given the high bandwidth networks available today. High TCP initial congestion windows, such as 10 segments can hold 14KB of data. Objects with size less than 30KB will be loaded in browsers within two RTT's. There have been few studies conducted about increasing the initial congestion window of TCP to a higher value. The discussions lead to an adaptive scheme [28] of increasing the initial congestion window and having a fixed value for initial congestion window.

If we take a high-speed network with high latency then the amount of data on the wire/medium is huge, whereas a high speed network with low latency has less data on wire/medium. Consider a HSPA network with 6 Mbps link speed and 100 ms RTT.

The Bandwidth Delay Product (BDP) will be $6 \times 10^6 b/s \times 10^{-1} s = 6 \times 10^5$ b or 75KB

Hence, 75KB is the amount of data present in network which is not acknowledged. When we use initial window of 3 segments the network is under-utilized even though the network can hold 75KB of data, when the window grows it will not reach the maximum threshold as the web transactions are generally short lived.

There are advantages as well as disadvantages of having higher initial congestion window. Advantages are reduction in latency, keeping up with the web object size in today's world, and support for the new loss recovery techniques like Limited Transmit [8] and Early Retransmit [7]. Reduction in latency is mainly achieved due to the reduction in RTT. Since the congestion window grows exponentially the number of RTT's will reduce. The web object size increases day by day. Transactions [9], such as e-mails, updates such as weather information benefits from higher initial congestion window. The beneficiaries of higher initial congestion window would be small sized objects. If the initial congestion window is 10 segments then the size on wire/medium is around 14KB. Objects with size equal to or less than 14KB will load in the initial burst itself, whereas initial window of 3 segments would require additional round trips to complete the same 14KB transaction.

The disadvantages can be on the individual connection level or network level. On an individual connection level, this can lead to an early congestion avoidance phase. This could happen if there is packet loss resulting from small buffers in routers. There can be cases where it would be better to start with initial congestion window of 3 instead of a higher number. The losses can also initiate retransmission timeouts. On the network level, large initial congestion window can lead to network congestion if packet loss occurs. The higher initial window can also lead to spurious RTO on low-bandwidth paths [9].

If we consider a perfect network with no congestion or packet loss then the TCP window grows exponentially. When the TCP window reaches the slow start threshold, TCP congestion window grows in additive increase or multiplicative increase method. The Figure 10 is an example of a perfect network with no packet loss, high slow start threshold value, and Maximum Segment Size (MSS) of 1460 bytes. We can see that, TCP initial congestion window starts at three segments and 4KB of data is transferred. When the ACK of three segments reach the sender, it increases the TCP sender window to six segments. This process goes on until it reaches the TCP slow start threshold. Hence, a high TCP initial congestion window can complete the data transfer in lesser number of round trips. When a TCP initial congestion window of 10 is used, 14KB of data is transferred in the initial burst. The effect of initial window of 3 and initial window of 10 is shown in the figure RTT and TCP *initcwnd*.

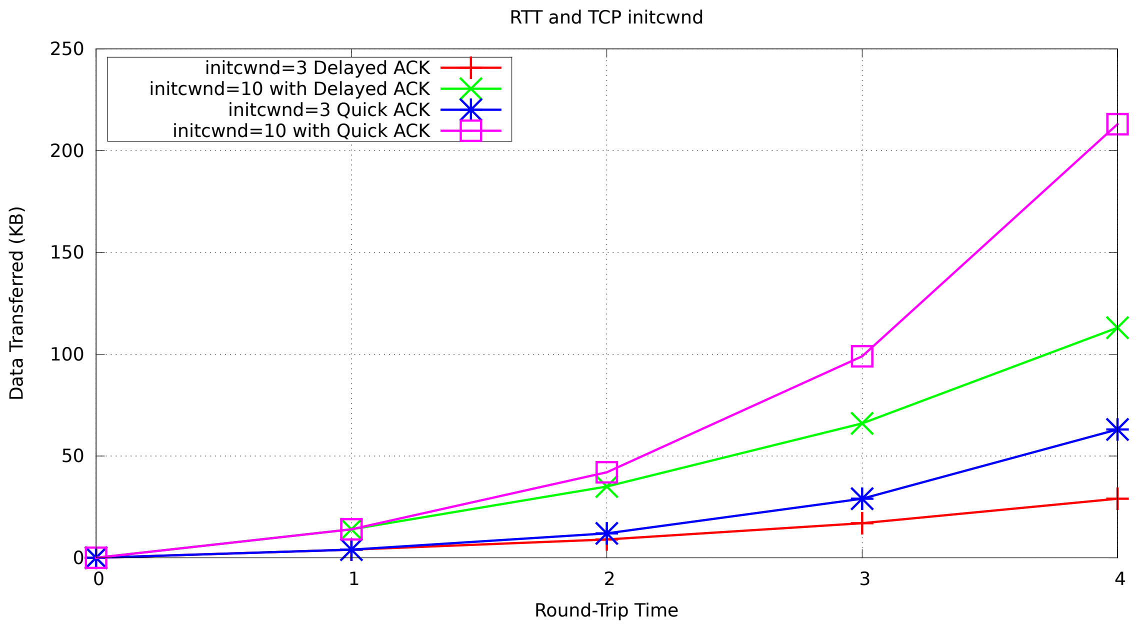

Figure 10: RTT and TCP *initcwnd*

The Figure 10, shows the behavior of TCP congestion window for every RTT. The X-axis represents the RTT and Y-axis represents the cumulative data transferred in KB. In TCP with delayed ACK, sender is sending ACK for every second segment increasing the congestion window in the sequence 3, 4, 6, 9, 13 for *initcwnd* of three. The sequence follows 10, 15, 22, 33, 49 for *initcwnd* of 10 whereas in quick ACK, the TCP congestion window doubles after each RTT. The sequence follows 3, 6, 12, 24, 48 for *initcwnd* of 3 and 10, 20, 40, 80, 160 for *initcwnd* of 10.

3.4.2 Receive Window

Increasing the initial congestion window is not sufficient to achieve maximum performance. We should make sure that receiver can accommodate the large initial windows TCP sender is using. TCP has a receive window on the client side which is advertised to the server in each packet server receives. The server can send only the minimum of the TCP congestion window and receive window. If the client's receive window is not large enough to accommodate the server's high initial congestion window value, the server has to use the smaller sender window value. For example, initial congestion window of 10 segments is equal to 14KB and receiver window size is 10KB for Linux as shown in Table 2. This is cannot happen today as latest Linux kernel versions (2.6 onwards) use auto tuning [54]. The receive window size depends on the OS distribution [16].

Initial *rwnd* Size	
OS	Avg. Size
Linux	10KB
FreeBSD	58KB
Win 7	41KB
Mac	270KB

Table 2: Initial Avg. *rwnd* Size [16]

From the Table 2 we can see that the initial *rwnd* of the Linux is comparatively small whereas Mac has the biggest initial *rwnd* value. Windows operating system claims to have a dynamic receive window size which varies from 8KB to 64KB [41] depending on the link speed.

3.5 Summary

In this section, we discussed about the shortcomings of current protocols. First part discussed the shortcomings in HTTP and TCP. The features available in HTTP and TCP, which are not used in current browsers, web servers, and proxies. Second part proposed possible solutions or features that can be included in HTTP and TCP. It also introduced the SPDY protocol proposed by Google and discussed the features of the protocol. The next section describes the test setup that was created to test the SPDY and TCP higher initial window.

4 HTTP, TCP, and SPDY Test Setup

4.1 Importance of Measurements

In this section we describe the test setup created to measure the performance of the SPDY protocol in comparison to HTTP protocol in cellular networks. The target was to analyze the impact on PLT, when using HTTP header compression, and multiplexing in cellular networks, which are known for high latency and low bandwidth conditions. The impact of an increased initial congestion window was another point of interest in cellular networks. We developed an automated framework that can measure the web performance parameters such as page load time, number of TCP connections, and size transfer distributions. The detailed description of the test setup is explained in the following sub-sections.

4.1.1 Test Bed and Setup

The test bed comprises a client (Google Chrome browser) and a server (customized web server from Google, named Flip) interconnected by cellular data network. There are a number of shell scripts running on both client and server sides which capture the packets using *tcpdump* and processes the packet dumps to generate the results and plots. The location of the test bed is NRC in Helsinki and Espoo, Finland. We chose the locations based on the availability and better reception of GPRS and HSPA networks. Most of the places in Finland do not have GPRS but EDGE. For our tests we consider GPRS and HSPA as we wanted to test the extreme conditions available in terms of bandwidth and technology.

4.1.2 Network Setup

The Figure 11 shows the detailed configuration of the test bed. The client side uses two Nokia N95 mobile modems connected to a PC using USB cables. The mobile phones are configured to be in either GPRS only mode or HSPA only mode. The mobile operator is a Finnish carrier, TeliaSonera. For accessing the LTE network, we use a LTE supported USB dongle. The client and server exchange tcpdumps and other test related info through a dedicated LAN connection between them. This second interface was required to exchange the tcpdumps without interrupting the cellular data network testing.

The access networks for the tests are GPRS and HSPA. The GPRS network has an average RTT of around 400 ms with a standard deviation of 30ms. The average RTT is calculated using a ping of 64 bytes to the server 30 times before the tests when the radio state is in IDLE. The downlink and uplink is 0.06 Mbps and 0.02 Mbps respectively, measured using the website www.speakeasy.net meant for calculating the bandwidth. The HSPA network has an average RTT around 100 ms with a standard deviation of 5 ms. The downlink and uplink is 3.0 Mbps and 0.37 Mbps respectively. The ISP is TeliaSonera in both access networks. The RTT

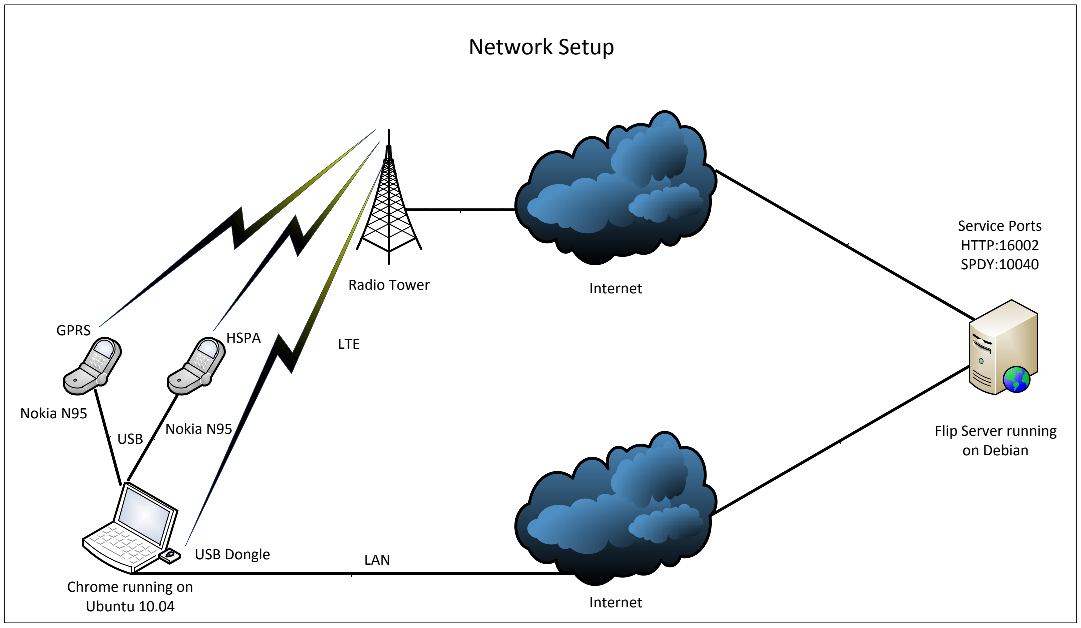

Figure 11: Network Setup

and bandwidth values can be different depending on the time of the day and data traffic. The Nokia N95 mobile phone specifications are given in the Table 3.

Nokia N95 Specification		
Technology	Uplink	Downlink
WCDMA 850/1900(HSDPA)	384 Kbps	3.6 Mbps
GSM/EDGE	118.4 Kbps	177.6 Kbps
EGPRS Class B	177.6 Kbps	296 Kbps

Table 3: Nokia N95 Specification

4.1.3 Client Side Configuration

The client is Google Chrome browser hosted on a Linux based system with Ubuntu distribution 10.4. The kernel version is 2.6.35.4. An updated version of the kernel from the stock is mandatory, as most of the Linux distributions below the kernel version 2.6.35.4 do not have provision for accommodating the TCP receiver window. TCP receiver window is a crucial part when conducting the tests with high TCP initial congestion window.

4.1.4 Server Side Configuration

The server is called Flip server and hosted on a PC running Linux (Debian) distribution. The Flip server can be built from the same Chromium repository [12] as

Google Chrome. The server hosts the content in a pre-defined format and particular website domains are divided as it is in the real world with sub-domains. We generated the database by visiting the desired websites to be tested using Chrome browser in record-mode and dumped the recorded files in encoded format. This particular procedure is Windows-specific. The Flip server can act as HTTP and SPDY server.

4.1.5 Changes From The Default Binaries

The binaries have been modified so that it fits to the test setup and scenarios. The Flip server is based on the spec 2 [34] of SPDY draft. Hence, changes made after the spec 2 release is not applicable to the results presented in this thesis.

4.1.6 Other Changes

One of the main exception in the test setup is the lack of DNS queries. In real world, DNS queries introduce a significant amount of delay in web performance. The delay can be from few milliseconds to several seconds. Since the server is hosted on a single static IP address and testing procedure demands direct queries to server IP address, DNS was omitted from the test setup.

4.1.7 Testing Methodology

The test work-flow is controlled by shell scripts running on both ends. The steps are given below.

Step 1 . The shell scripts on the client invokes the Chrome browser, Flip server, and the *tcpdump* on the client and the server network interfaces.

Step 2 . The shell scripts sleep for a particular amount of time to make sure that there is no data transfer during that time and radio state is in IDLE. This procedure is repeated before every test.

Step 3 . The desired website is requested in a Chrome browser by invoking the desired command line switches.

Step 4 . The packets are captured during the whole testing process and stored in MySQL database for batch processing. At the end of the tests, the *tcpdump* data is processed for generating desired statistics.

Step 5 . A native C program, developed for analyzing *tcpdump* is used to measure the Page Load Time (PLT). When the webpage is loaded, *tcpdump* is stopped.

Step 6 . The steps 1, 2, 3, 4, and 5 are repeated for different test cases.

We measure the start time by checking the timestamp of the first SYN packet sent from client and to measure the end time, we measure the timestamp of the last packet to arrive at the client from the server before an idle period of 10 seconds. The idle period is assumed to be the end of webpage load and the packet to arrive before the idle time is considered the last packet.

4.2 Test Cases

Various test cases were developed to analyze the HTTP protocol improvements. First is the comparison between SPDY and HTTP. Second is the increased TCP initial congestion window. Third test case is the 200ms artificial delay between server and client to simulate the real world mobile browsing scenario where RTT is high. Fourth is the SYN to SYN-ACK delay, measured in GPRS, HSPA and LTE networks. In the original test setup, the latency in GPRS and HSPA network is mainly channel transition delay and radio network latency. In the real world, depending on the location of server there is a delay which ranges from 10ms to 400ms.

4.2.1 Test Cases in Detail

We consider a blend of SPDY and HTTP protocol variants. The aim is to measure the gain in Page Load Time (PLT) when using SPDY, whereas we are also analyzing the effect of large initial congestion window and number of TCP connections per domain. We are evaluating the following test cases in GPRS, HSPA and LTE networks. All of these four protocol variants are tested over GPRS, HSPA, and LTE networks, which make 16 test cases.

Test Cases			
Network	initcwnd	Protocol	Max. Parallel TCP connections
GPRS	3	HTTP	6 per domain
	3	SPDY	1 per domain
	10	HTTP	6 per domain
	10	SPDY	1 per domain
HSPA	3	HTTP	6 per domain
	3	SPDY	1 per domain
	10	HTTP	6 per domain
	10	SPDY	1 per domain
LTE	3	HTTP	6 per domain
	3	SPDY	1 per domain
	10	HTTP	6 per domain
	10	SPDY	1 per domain
LTE - 200 ms	3	HTTP	6 per domain
	3	SPDY	1 per domain
	10	HTTP	6 per domain
	10	SPDY	1 per domain

Table 4: Test Cases

The SYN to SYN-ACK delay test cases were introduced to find the radio bearer setup delays in GPRS, HSPA and LTE networks. When a SYN packet is sent, it wakes up the radio and makes it move from IDLE state to FACH or DCH in HSPA network. In GPRS and LTE network, the state transitions are not significant as they are in CONNECTED state almost all the time. We measured the SYN to SYN-ACK delay in three different HSPA networks in Finland, named DNA, Elisa, and Sonera.

Operators	**GPRS**	**HSPA**	**LTE**
Sonera	✓	✓	✓
Elisa	×	✓	×
DNA	×	✓	×

Table 5: SYN to SYN-ACK Delay Measured in Different Finnish Operators

4.2.2 The Reason for Choosing Different Test Cases

We wanted to know how header compression, multiplexing helps in HTTP. Hence, we chose SPDY as the competitive protocol. Apart from improving the way HTTP behaves, our interest spanned over TCP initial congestion window. This lead to the test cases with TCP initial window of 3 segments and TCP initial window of 10 segments.

4.2.3 Websites and Properties

Website	Domains	Objects	Website	Domains	Objects
Baidu.com	2	7	Cnet.com	22	160
Bing.com	5	16	Microsoft.com	14	78
Amazon.com	8	82	Spotify.com	6	27
Craiglist.org	4	8	Nytimes.com	20	140
Ebay.com	12	38	Qq.com	16	86
Facebook.com	3	15	Iltalehti.fi	8	204
Imdb.com	13	82	Hs.fi	20	226
Kernel.org	1	15	Yle.fi	7	64
Linkedin.com	7	15	Mtv3.fi	8	120
Megatuutti.fi	1	2	Aol.com	15	75
Ovi.com	4	39	Cnn.com	12	125
Wikipedia.org	5	16	Espn.com	18	88
Wordpress.com	13	48	Tumblr.com	4	21
Yahoo.com	8	54	Bbc.co.uk	11	65
Youtube.com	8	22	Ask.com	10	34

Table 6: Website and Properties

The table 6 shows the websites we used for testing. First column is the names of websites tested. Second column is the number of sub-domains in each website.

This is mainly to verify the nature of SPDY regarding the single TCP connection per domain policy. Last column gives information about the number of objects in each website. The size of the website on disk cannot be calculated accurately as the web pages are stored in encoded format.

4.3 Summary

In this section we discussed the test setup for the measurement and the specification of the access networks and devices used for the measurements. This section also discusses about the various test cases and the reason for selecting them. Apart from that, this section describes the properties of the websites used for testing. In the next section we will analyze the results obtained from the measurement studies.

5 HTTP, TCP and SPDY Results

In this section, we present the data and results gathered from our tests. We discuss the SYN to SYN-ACK round-trip time in radio networks, payload and header compression, the gain in PLT when using SPDY, the reduction in number of TCP connection per domain, and the effect of initial congestion window in cellular networks. The results presented are based on more than 100 iterations for each protocol.

5.1 SYN to SYN-ACK Round-Trip Time Measurements

The establishment of a TCP connection is characterized by a Three-way handshake. The client sends a SYN packet to the server, server responds with a SYN-ACK packet and client completes the Three-way handshake by sending an ACK packet. In HSPA network, the radio bearer should be in FACH or DCH state to send or receive any packets. Since the test setup lacks the DNS, the radio is in IDLE state at the beginning of the tests. The first packet sent to the network is the SYN packet from the client. We measured the time duration between sending a SYN packet and recieving a SYN-ACK packet. This time or delay roughly estimates to the radio state transition delay and network delay in cellular data network. The measurement study was performed on three major mobile phone operators in Finland namely Sonera, Elisa, and DNA. These measurements have been performed using Nokia-N95 device hence there can be alterations in the radio state transition delays as the newer devices employs better technologies to reduce the radio state transition delays.

From the Figure 12 we can see that, all the three operators have more than 1.5 seconds delay between SYN and SYN-ACK exchange in HSPA network. In case of Elisa the delay goes well beyond 2 seconds whereas Sonera and DNA networks seems to be in the range of 1.5 to 2 seconds range. One important fact to understand from this measurement is that we can reduce this delay by sending a DNS query or any relevant packets to the network to wake up the radio network. Such pre-activation of radio network would remove the state transition delay from the above-mentioned SYN to SYN-ACK round-trip delay. Another important fact to be considered is the effect of this delay on page load time. The elimination of such delay can give 1.5-2 seconds reduction in page load time of websites.

The next step involved measuring the similar delay in GPRS, HSPA, and LTE networks. We measured the SYN to SYN-ACK round-trip delay in Sonera network. HSPA round-trip delay values were taken from the earlier measurements and GPRS round-trip delay values fell between 0.5 second to 1 second. This is an indication that the timers of radio state transition delay in GPRS is smaller compared to HSPA networks. Hence, the reduction of SYN to SYN-ACK round-trip time in GPRS network would fetch the pages 0.5 second to 1 second earlier than the actual page load time. In the LTE network the delay is comparatively small and the numerical

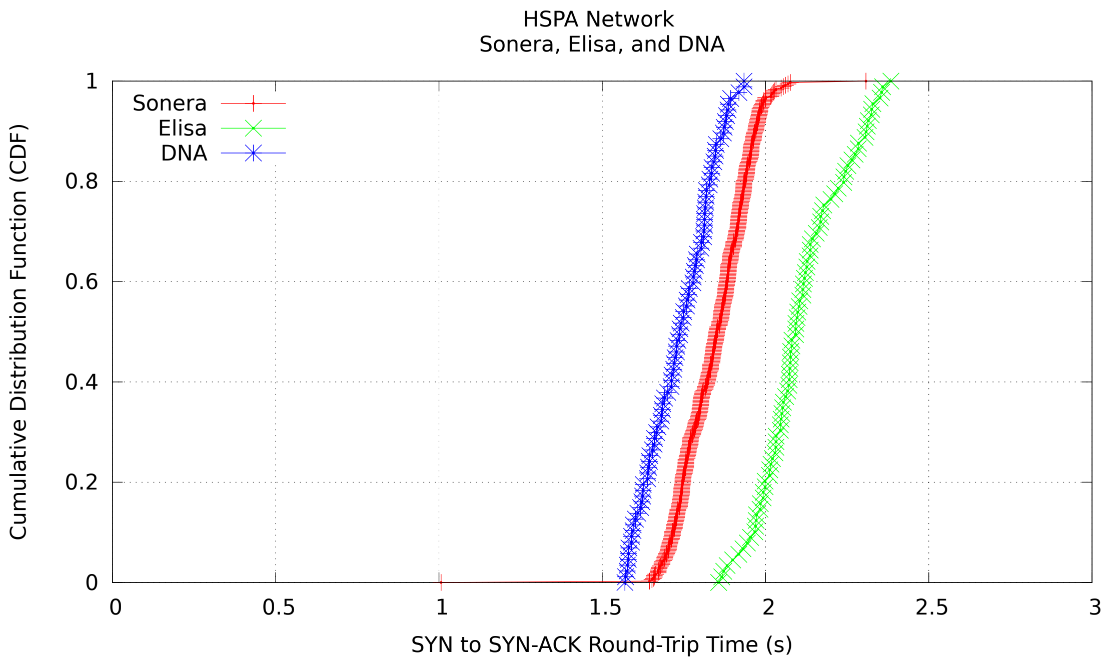

Figure 12: SYN to SYN-ACK Round-Trip Time : Three Mobile Operators in Finland

values are below 100 ms. The distribution of delay in GPRS, HSPA, and LTE network is shown in the Figure 13.

5.2 Size of Websites using HTTP and SPDY Protocols

There is a significant reduction in total payload when using SPDY. The Figure 14 shows the percentage reduction in payload using SPDY in comparison to HTTP. The X-axis represents the websites tested and Y-axis represents the percentage reduction in total payload when using SPDY. SPDY employs only the header compression hence the total payload compression over SPDY is the result of request and response header compression. We can see that total payload compression ranges from 0.47% to 11.86%. The average compression over 30 websites is 4.59%.

From the Figure 15, we can see the request header compression in SPDY. The request header compression is calculated by comparing the HTTP and SPDY GET requests and finding the size difference between them. We can see that the request header compression in all the cases is above 50%. High request header compression can help in networks where the uplink is low. In those cases, the browser will be able to make the GET requests size small. In some cases, the request header compression is above 80%. The average request header compression over 30 websites is 72.19%.

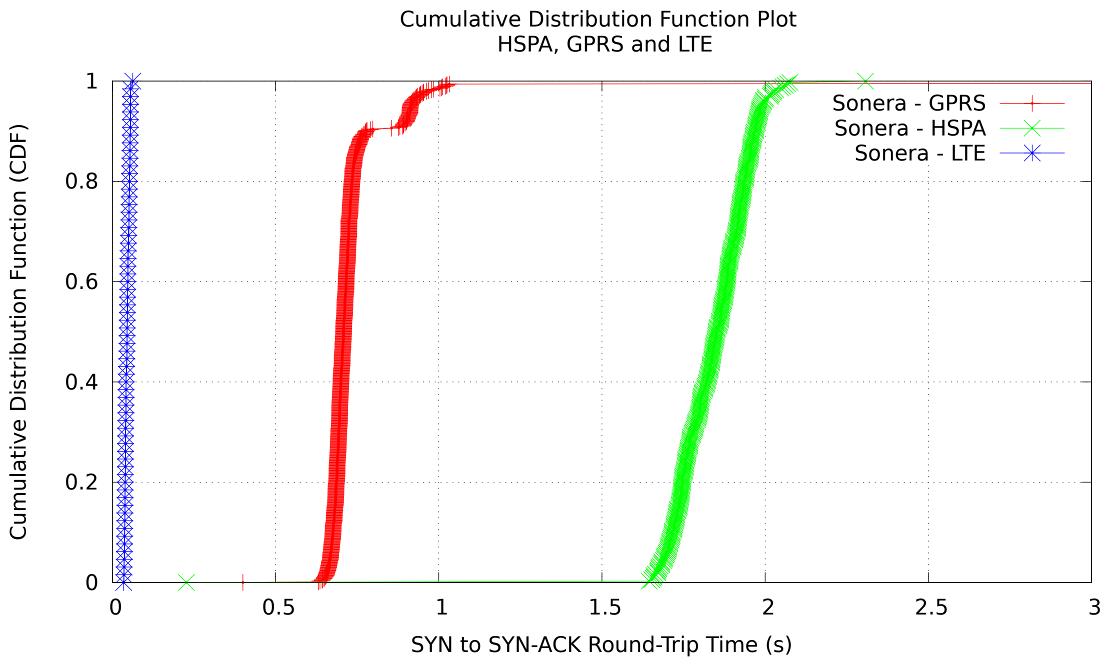

Figure 13: SYN to SYN-ACK Round-Trip Time : Sonera GPRS and HSPA Networks

The Table 7 presents the website, amount of data transferred using HTTP and SPDY, and the percentage gain in size using SPDY compared to HTTP.

5.3 HTTP, SPDY, and TCP Results on GPRS and HSPA Networks

This sub-section covers the HTTP and SPDY comparison results in GPRS and HSPA networks. The page load time is the main criteria in all of the tests. Two types of plots are presented for every website tested. The first plot is the page load time plot, which represents page load time for every website and compares it with different protocol variants. Second one shows the amount of bytes transferred over time for HTTP and SPDY protocols.

In case of page load time plots, the elements in the X-axis represent the protocol variants. HTTP-iw3 implies the HTTP protocol with a TCP initial congestion window of 3 segments. A similar naming convention is used throughout this document. The Y-axis represents the page load time in seconds. The quartile samples are plotted with the middle points as median.

We have measured the number of TCP connections used by each protocol. The number of TCP connections is the number of successful TCP connection establishments made. In case of SPDY, some cases might show more number of TCP

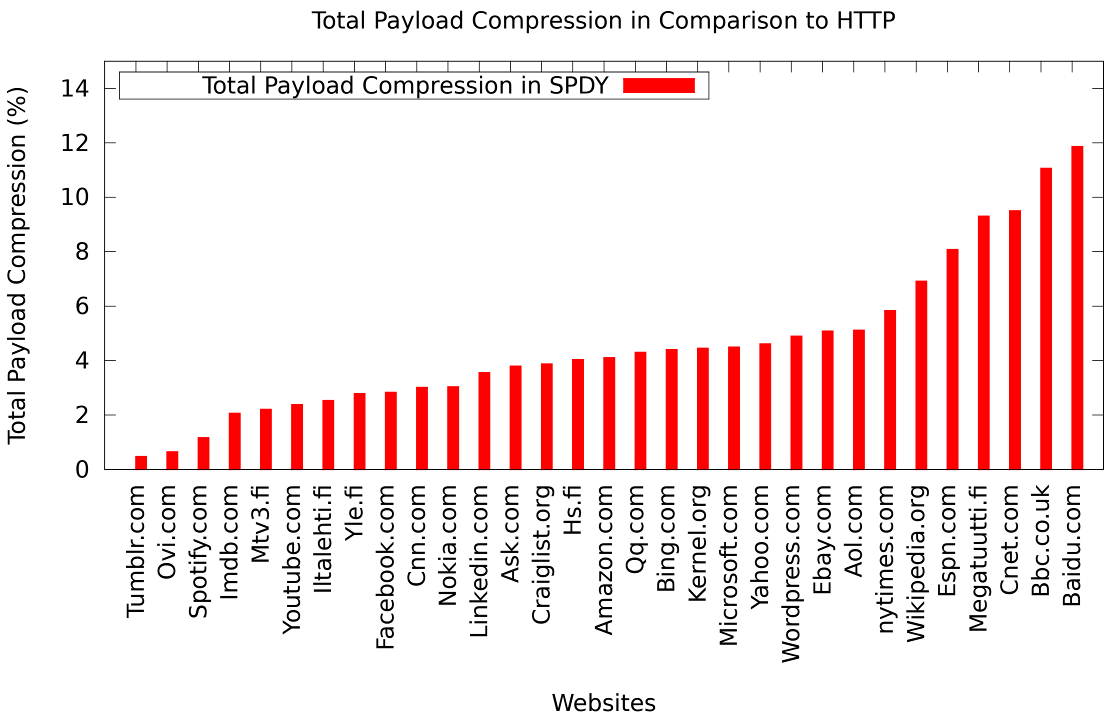

Figure 14: HTTP vs. SPDY Payload Compression

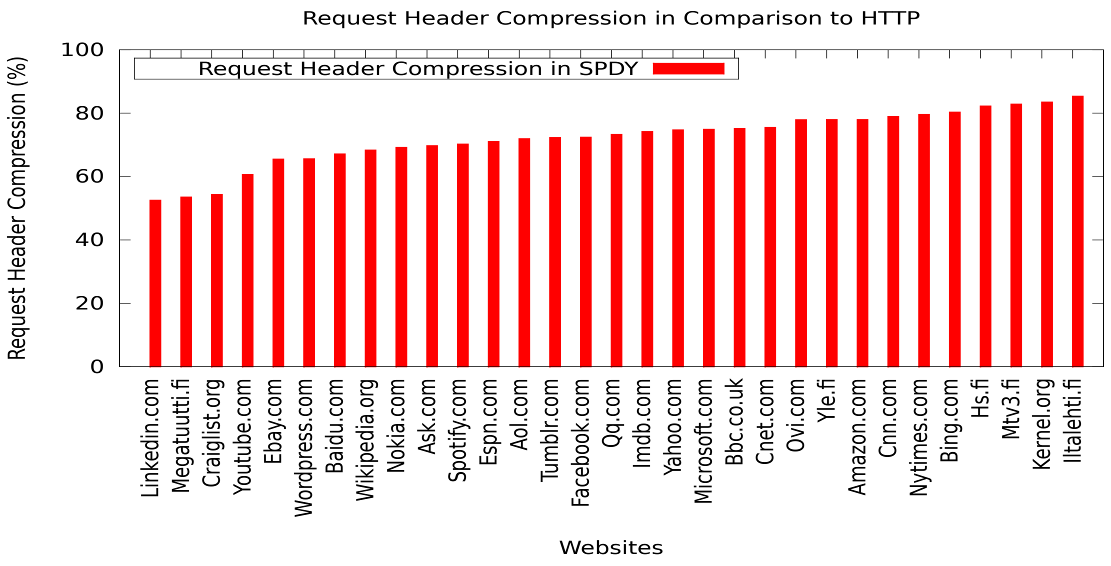

Figure 15: Request Header Compression

connection than it is supposed to use. SPDY uses the SSL handshake for checking the SPDY compatibility. Since the test setup uses the SPDY version without the SSL negotiation, the browser is not aware of the SPDY protocol in the beginning of

Website	Size (HTTP)	Size (SPDY)	Size Difference
Baidu.com	11.8KB	10.4KB	13.4%
Bing.com	100KB	95.6KB	4.6%
Amazon.com	564KB	540.9KB	4.2%
Craiglist.org	41.3KB	39.7KB	4.0%
Ebay.com	279.5KB	265.3KB	5.3%
Facebook.com	144.7KB	140.6KB	2.9%
Imdb.com	1226.8KB	1201.5KB	2.1%
Kernel.org	85.3KB	81.5KB	4.6%
Linkedin.com	121KB	116.7KB	3.6%
Megatuutti.fi	4.3KB	3.9KB	10.2%
Ovi.com	1599.2KB	1588.9KB	0.6%
Wikipedia.org	94KB	87.5KB	7.4%
Wordpress.com	386.6KB	367.7KB	5.1%
Yahoo.com	425.1KB	405.5KB	4.8%
Youtube.com	227KB	221.6KB	2.4%
Cnet.com	842.2KB	762.2KB	10.4%
Microsoft.com	681KB	650.4KB	4.7%
Spotify.com	807.9KB	798.5KB	1.1%
Nytimes.com	977.2KB	920.2KB	6.1%
Qq.com	472KB	451.7KB	4.4%
Iltalehti.fi	2822.1KB	2750.7KB	2.5%
Hs.fi	1680.1KB	1612.4KB	4.1%
Yle.fi	802.8KB	780.5KB	2.8%
Mtv3.fi	1561.8KB	1527.4KB	2.2%
Aol.com	682.4KB	647.5KB	5.3%
Cnn.com	905.7KB	878.4KB	3.1%
Espn.com	800KB	735.4KB	8.7%
Tumblr.com	1222.4KB	1216.7KB	0.4%
Bbc.co.uk	638.2KB	567.6KB	12.4%
Nokia.com	115.7KB	112.2KB	3.1%
Ask.com	245.5KB	236.2KB	3.9%

Table 7: HTTP and SPDY Transfer Size

the transaction. Hence it makes many number of TCP connections considering the bandwidth and delay parameters.

The second plot represents the amount of bytes transferred over time for the initial congestion window of 3 and 10. X-axis represents the time in seconds whereas Y-axis represents the amount of bytes transferred. The data over HTTP is marked with dark color line with star symbols and data over SPDY is marked with light color line with plus symbols. Every line has two symbols; the triangle on every line represents the TCP SYN packets whereas the circle represents the HTTP GET requests. The transfer rate plot helps us understand when the SYN packets and HTTP GET requests are sent.

5.3.1 Test Case - Amazon.com

Taking the example of amazon.com, which has 8 domains and 82 objects. The size of the website over HTTP is 564KB and size over SPDY is 540.9KB. We can see the page load time plot for GPRS and HSPA networks, and transfer rate plot for initial congestion window of 3 and 10.

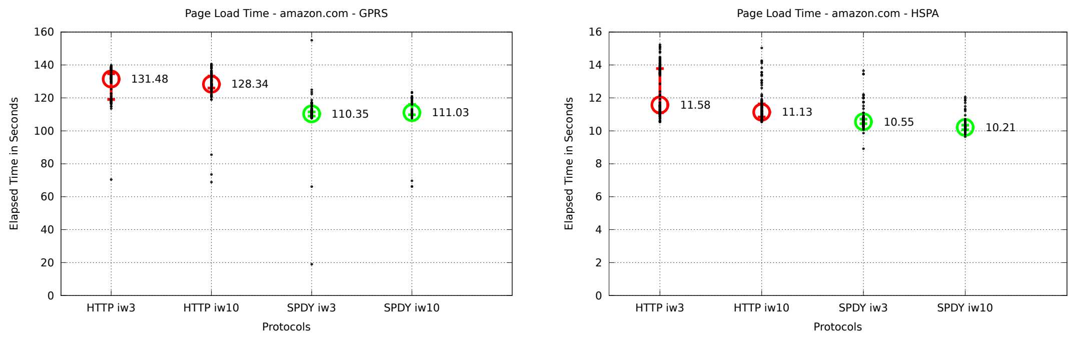

Figure 16: PLT in GPRS and HSPA Network for Amazon.com

From the Figure 16, it can be seen that in GPRS network, page load time of amazon over SPDY is 110.35 seconds whereas over HTTP it is 131.48 seconds. SPDY saves around 21 seconds in GPRS network. From the Figure 17, we can observe the size reduction due to header compression is the difference between the lengths of each protocol lines in the graph. The analysis of tcpdump shows that the number of TCP connections used by SPDY for amazon is is only 8. Amazon has 8 domains and with the help of multiplexing SPDY uses those 8 connections to fetch the objects from server. In HSPA network, it can be seen that the page load time of amazon over SPDY is 10.55 seconds whereas over HTTP it is 11.58 seconds. SPDY saves around 1 second in HSPA network. When we count the number of TCP connections, the SPDY variant uses the same 8 TCP connections.

From the Figure 18, we can also see that SPDY makes all the requests in the beginning of the transfer itself which is provisioned by the multiplexing feature, whereas HTTP makes the requests in a sequential order. The number of TCP connection used by HTTP is 28 but the number of TCP connections used effectively to transfer data is 22. The effect of initial congestion window is not seen in amazon, as the size of website is more than 500KB.

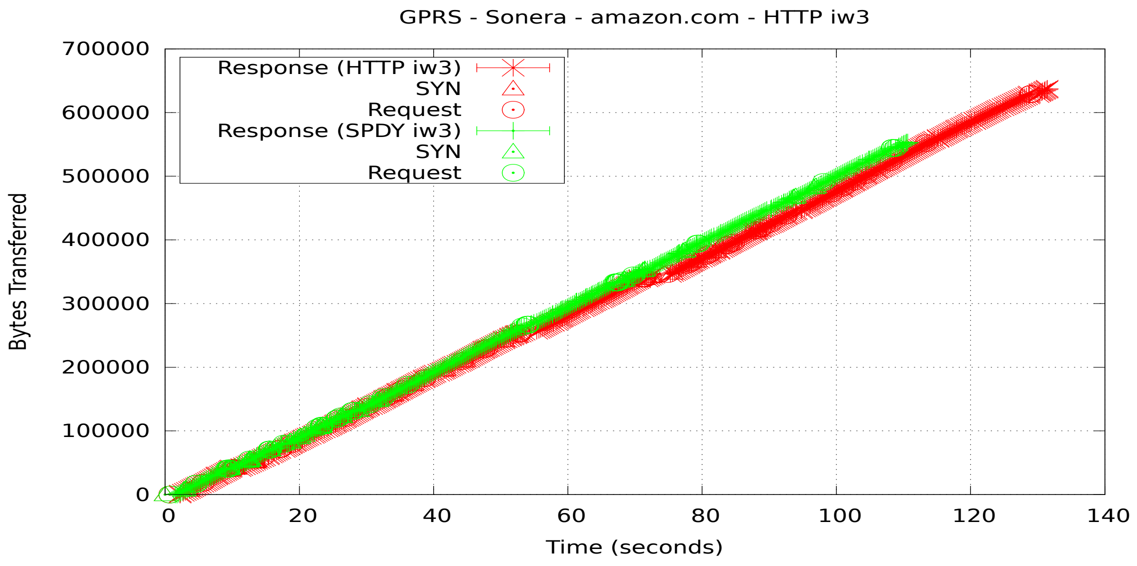

Figure 17: Transfer Rate Plot for initcwnd of 3 in GPRS

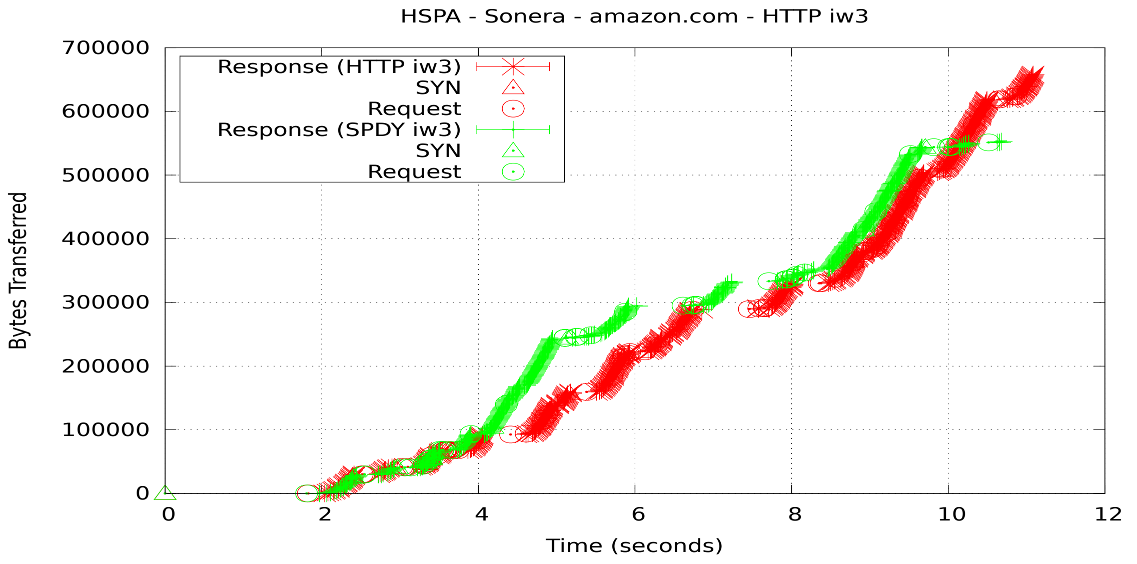

Figure 18: Transfer Rate Plot for initcwnd of 3 in HSPA

5.3.2 Test Case - Facebook.com

Taking the example of facebook.com, which has 3 domains and 15 objects. The size of the website over HTTP is 144.7KB and size over SPDY is 140.6KB. We can see the page load time plot for GPRS and HSPA networks and transfer rate plot for initial congestion window of 3 and 10.

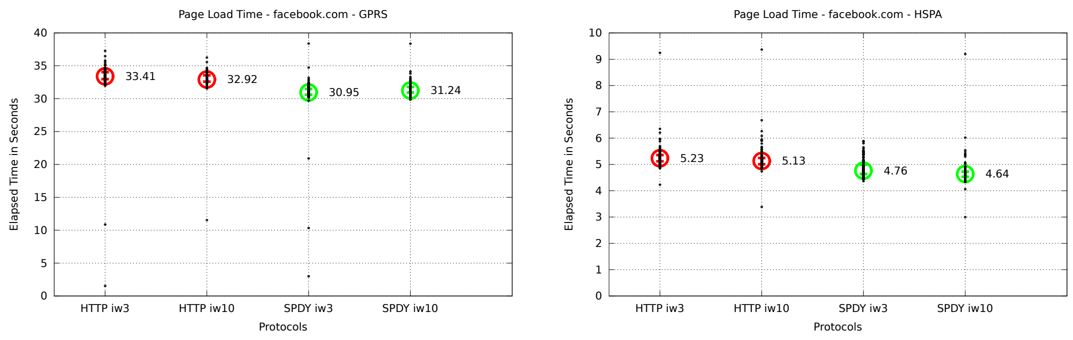

Figure 19: PLT in GPRS and HSPA Network for Facebook.com

From the Figure 19, it is clear that the page load time in GPRS network for facebook using HTTP is 33.41 whereas SPDY gains 2.46 seconds. The initial congestion window does not seem to have any influence in the case of facebook. Number of TCP connection used by facebook for HTTP is 9 and SPDY is 3. In HSPA networks, the page load time for facebook using HTTP is 5.23 and the gain using SPDY is 0.47 seconds. The initial congestion window does not seem to have much influence in the case of facebook. Number of TCP connection used by facebook for HTTP is 7 and SPDY is 3.

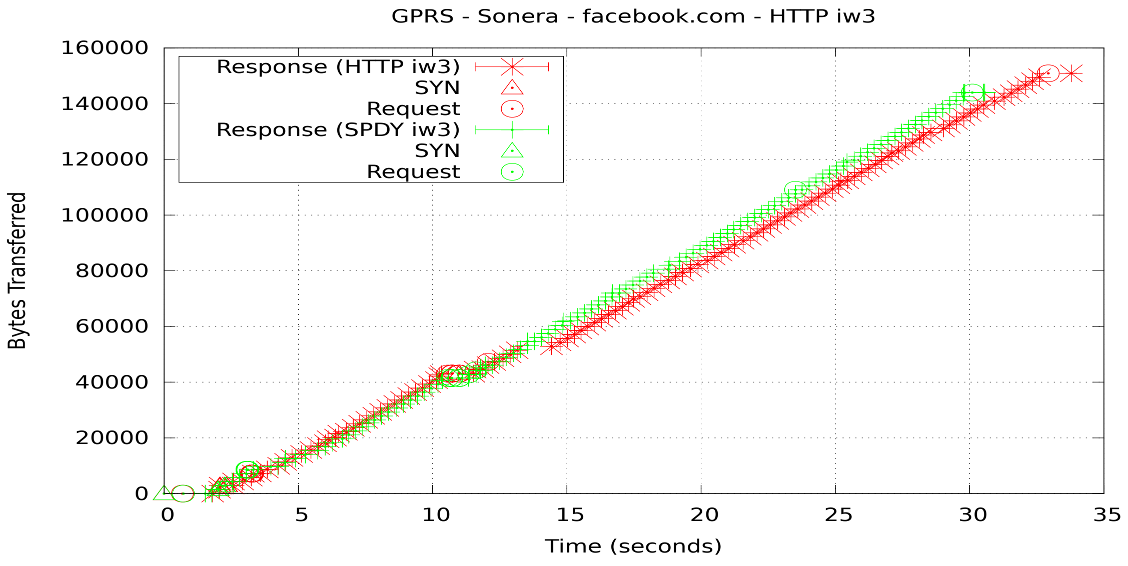

Figure 20: Transfer Rate Plot for initcwnd of 3 in GPRS

From the Figure 20, it is clearly visible that facebook does not gain from multiplexing. The difference between the page load times is clearly emerging from the header compression, which is close to 4KB. SPDY uses 3 TCP connections to transfer the website and open 5 connections in the beginning unaware of the SPDY protocol. HTTP opens 9 TCP connections and uses 8 of them for data transfer. Higher initial congestion window seems to have no influence in the case of facebook.com.

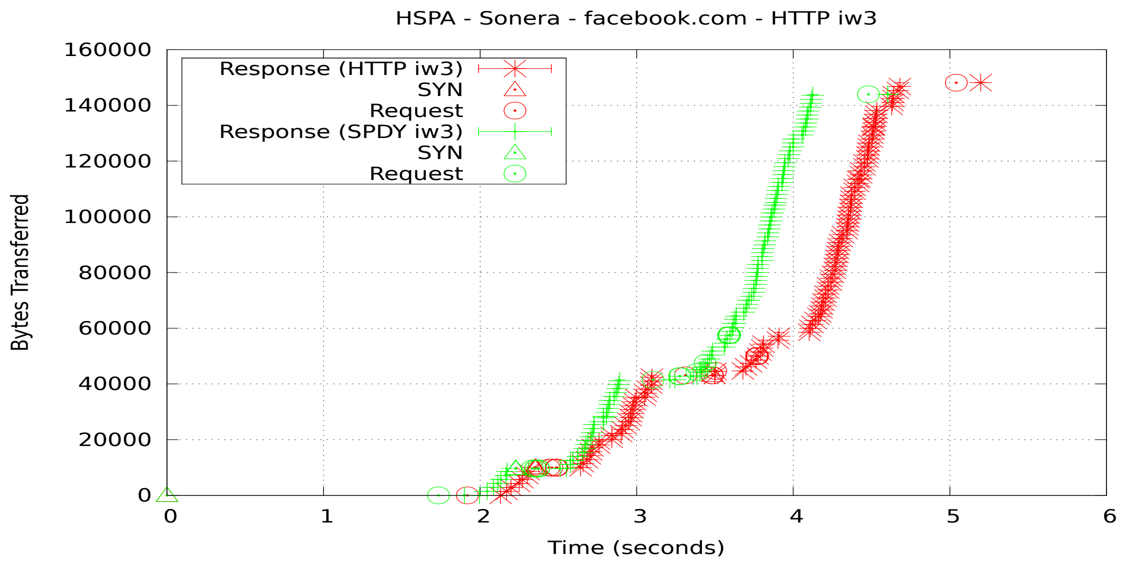

Figure 21: Transfer Rate Plot for initcwnd of 3 in HSPA

From the Figure 21, it is visible that SPDY gains much from multiplexing. In the HTTP curve, we can observe the period of 3 to 4 seconds clearly brings the difference between HTTP and SPDY. SPDY makes many HTTP GET requests during that time and overtake HTTP. Such cases are seen in other websites as well. SPDY uses 3 TCP connections to transfer the website and open 6 connections in the beginning unaware of the SPDY protocol. HTTP opens 7 TCP connections and uses 6 of them for data transfer. Higher initial congestion window seems to have no influence in the case of facebook.com.

5.3.3 Test Case - Baidu.com

Taking the example of baidu.com, which has 2 domains and 7 objects. The size of the website over HTTP is 11.8KB and size over SPDY is 10.4KB. We can see the page load time plot in GPRS and HSPA networks, and transfer rate plot for initial congestion window of 3 and 10.

From the Figure 22, we can see that in GPRS network the page load time for HTTP is 7.02 seconds whereas SPDY loads the page in 4.73 seconds. This is a considerable improvement in general and compared to the size of the website. Number of TCP connections used in HTTP is 6 and 2 in SPDY protocol. In HSPA network, we can see that the page load time for HTTP is 3.22 seconds whereas SPDY loads

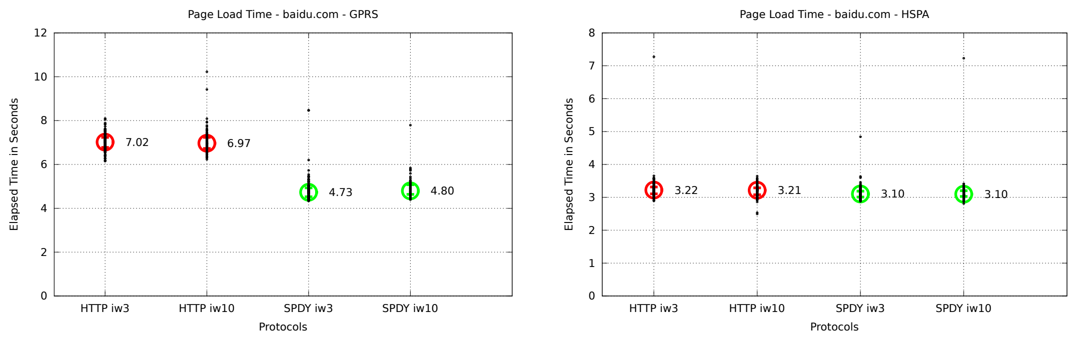

Figure 22: PLT in GPRS and HSPA Network for Baidu.com

the page in 3.10 seconds. This is not a noticeable improvement, as the user cannot perceive the gain in few milliseconds. Number of TCP connections used is 4 for HTTP and 2 for SPDY protocol.

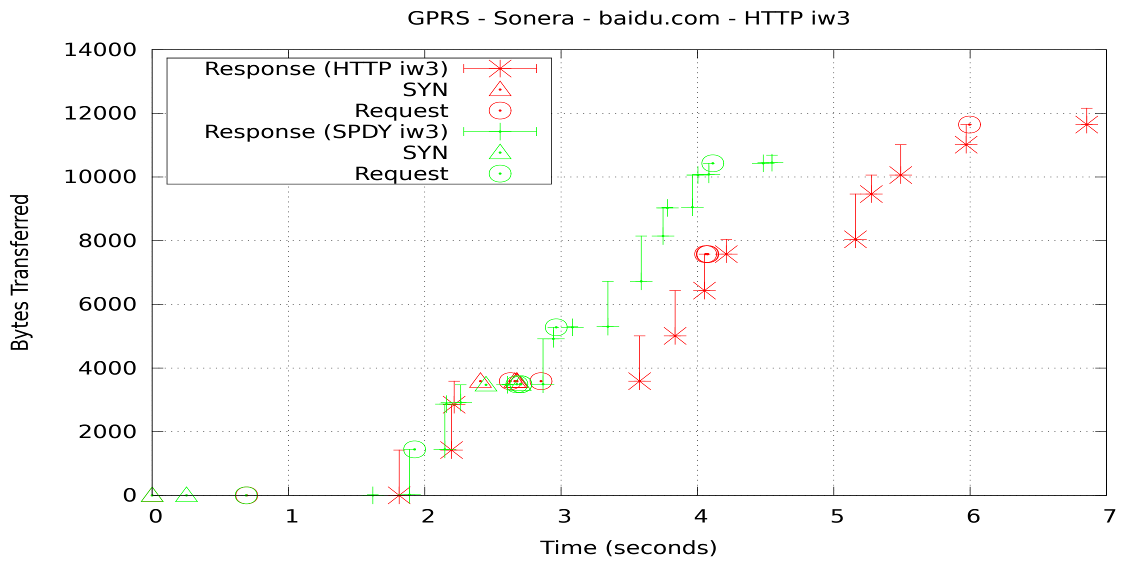

Figure 23: Transfer Rate Plot for initcwnd of 3 in GPRS

From the Figure 23, we can see that the multiplexing feature helps SPDY here. Around the time line of 3rd second SPDY makes 3 HTTP GET requests and completes the page load faster than the HTTP counterpart. Header compression seems to be very minimal which is 1.4KB. The number of TCP connection effectively used by SPDY is 2. HTTP opens 6 TCP connections and uses 4 TCP connections for data transfer. Higher initial congestion window is not effective in this case.

From the Figure 24, we can see that the effect of header compression is minimal. Multiplexing helps the SPDY protocol to issue the request using the existing TCP

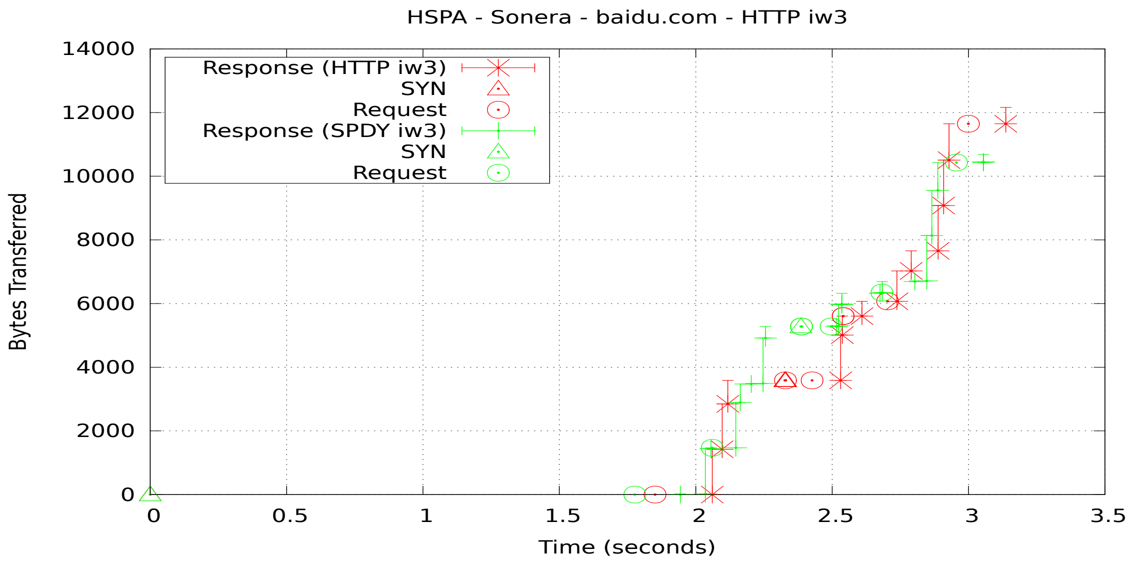

Figure 24: Transfer Rate Plot for initcwnd of 3 in HSPA

connection and save the round trip time to establish the TCP connection. The number of TCP connection effectively used by SPDY is 2. HTTP opens 4 TCP connections and uses 4 TCP connections for data transfer. Higher initial congestion window is not effective in this case. The SYN to SYN-ACK delay is above 1.5 seconds in this case as well. The reduction of this delay can make the PLT values lesser by 1.5 seconds.

5.3.4 Test Case - Bing.com

Taking the example of bing.com, which has 5 domains and 7 objects. The size of the website over HTTP is 100KB and size over SPDY is 95.6KB. We can see the page load time plot in GPRS and HSPA networks, and transfer rate plot for initial congestion window of 3 and 10.

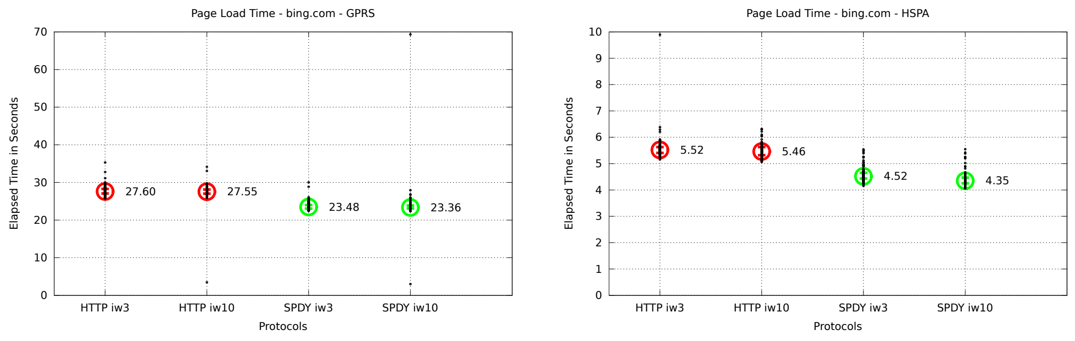

Figure 25: PLT in GPRS and HSPA Network for Bing.com

From the Figure 25, we can see that in GPRS network, HTTP takes 27.60 seconds to load whereas SPDY saves around 4 seconds. The size of website is around 100 KB and header compression is 4.4KB. Number of TCP connections used by HTTP is 12 whereas SPDY uses 5 connections. In HSPA network, we can see that HTTP takes 5.52 seconds to load where as SPDY saves 1 second. Number of TCP connections used by HTTP is 8 whereas SPDY uses 5 connections.

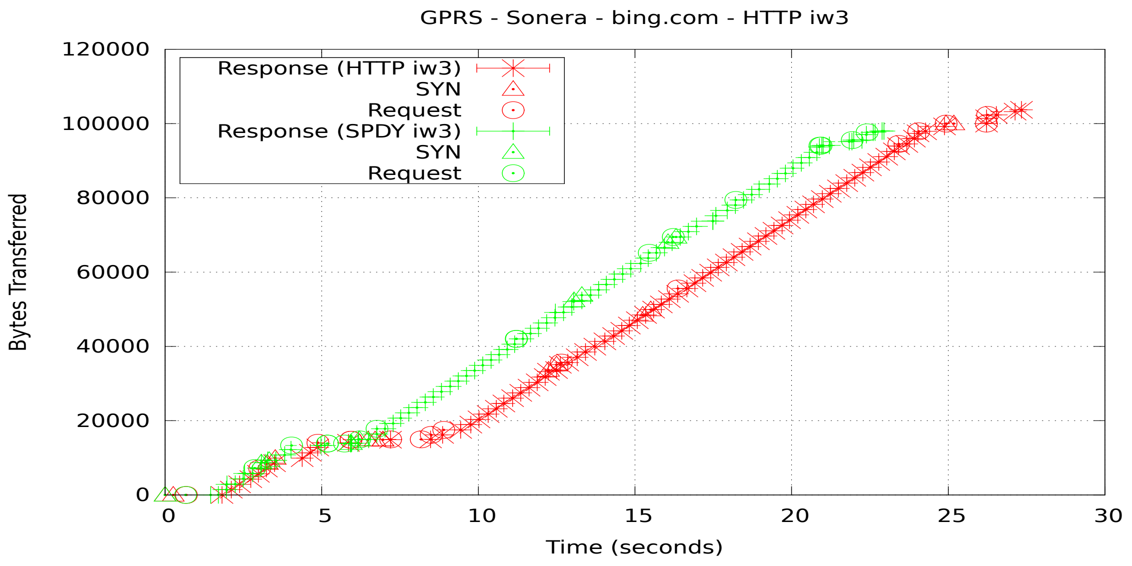

Figure 26: Transfer Rate Plot for initcwnd of 3 in GPRS

From the Figure 26, we can see that the difference in performance starts when HTTP tries to open TCP connections and waits for the connections establishment where as SPDY sends out HTTP GET requests on the same TCP connection, utilizing the multiplexing feature. SPDY uses only 5 TCP connections for effective data transfer. In case of HTTP, 12 TCP connections are opened and only 8 connections are used for actual data transfer. Higher initial congestion window seems to have no effect in this case as well.

From the Figure 27, we can see that the difference in performance starts when HTTP tries to open TCP connections and waits for the connections establishment whereas SPDY sends out HTTP GET requests on the same TCP connection, utilizing the multiplexing feature. Header compression is minimal in this case. SPDY opens 5 TCP connections and uses 5 TCP connections for effective data transfer. In case of HTTP, 8 TCP connections are opened and only 7 connections are used for actual data transfer. Higher initial congestion window seems to have no effect in this case as well.

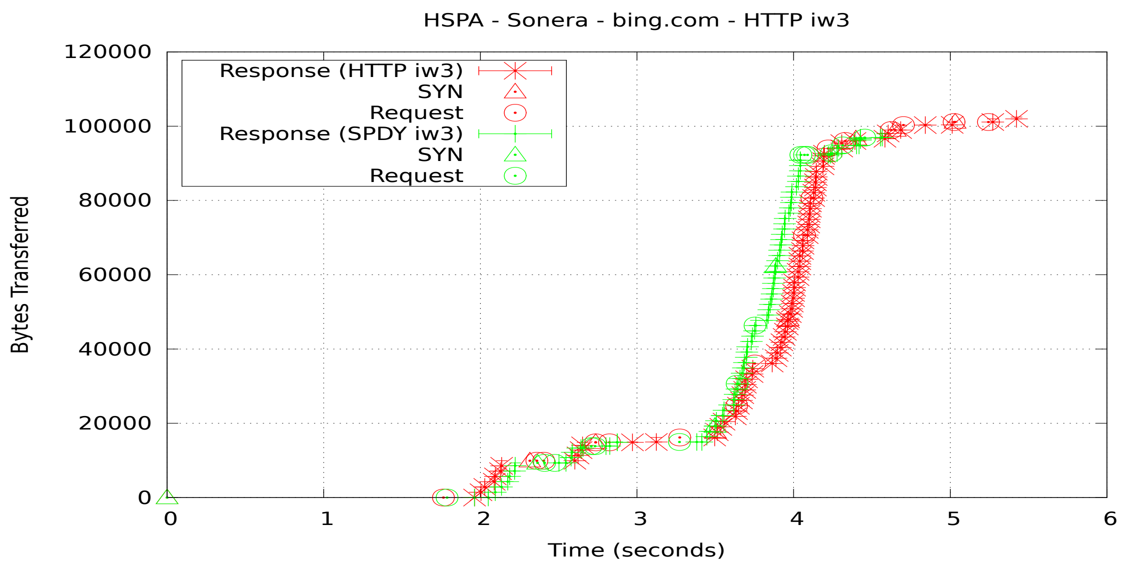

Figure 27: Transfer Rate Plot for initcwnd of 3 in HSPA

5.4 Summary of Results in GPRS

The Figure 28 compares the gain over HTTP *initcwnd* 3 by other protocol variants. The X-axis represents the % gain over HTTP *initcwnd* 3. All the three variants HTTP *initcwnd* 10, SPDY *initcwnd* 3 and SPDY *initcwnd* 10 are show in the Figure 28. We can see that HTTP with *initcwnd* 10 shows negative effect in few websites and do not show any improvement over many websites. HTTP with *initcwnd* 10 shows maximum improvement of 6% in one of the websites. SPDY with *initcwnd* 3 and SPDY with *initcwnd* 10 performs almost equal. There is no significant advantage in using *initcwnd* 10 in SPDY. Hence using SPDY in GPRS network can be beneficial, whereas the *initcwnd* 10 is not helping in GPRS network.

5.5 Summary of Results in HSPA

The Figure 29 compares the gain over HTTP *initcwnd* 3 in HSPA network. HTTP with *initcwnd* 10 shows improvement close to 7% in one of the websites. Unlike in GPRS network, HSPA network benefits slightly with *initcwnd* 10. When we look at the SPDY performance, both SPDY with *initcwnd* 3 and SPDY with *initcwnd* 10 performs better than HTTP with *initcwnd* 3. The improvements are in positive ranges, reaching close to 20% in some websites. Hence using SPDY in HSPA network will lead to better page load time of websites and use of *initcwnd* 10 can be beneficial in some scenarios.

One important fact to notice here is the SYN to SYN-ACK delay. Even though first SYN is sent at 0 seconds, the first HTTP GET request is made approximately

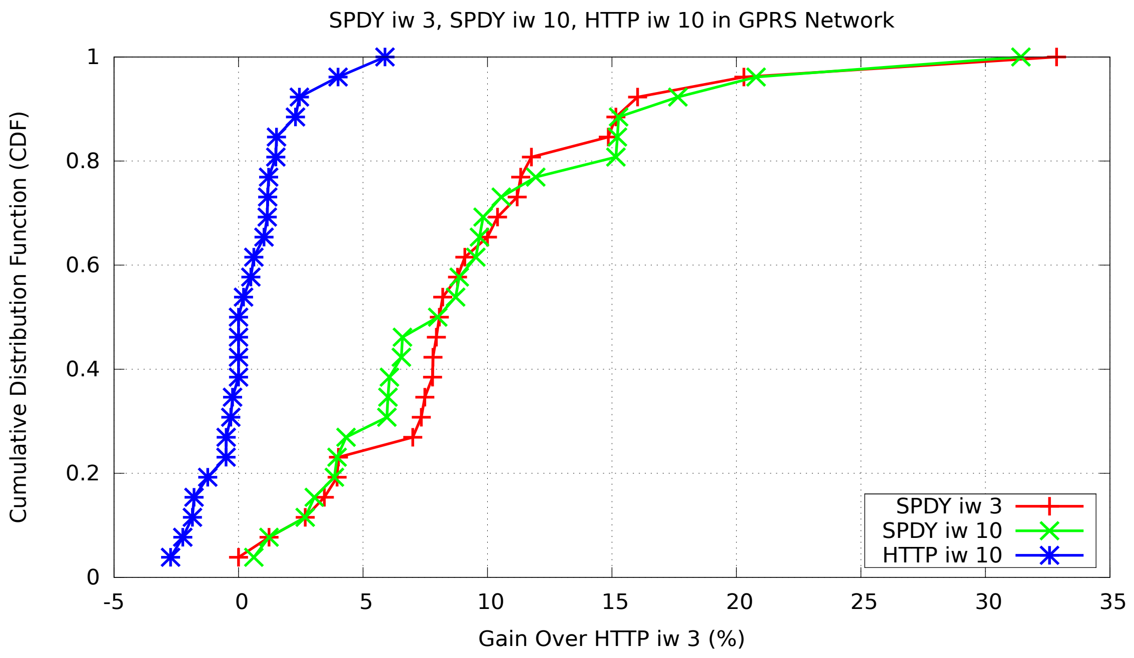

Figure 28: GPRS, Comparing HTTP and SPDY Variants

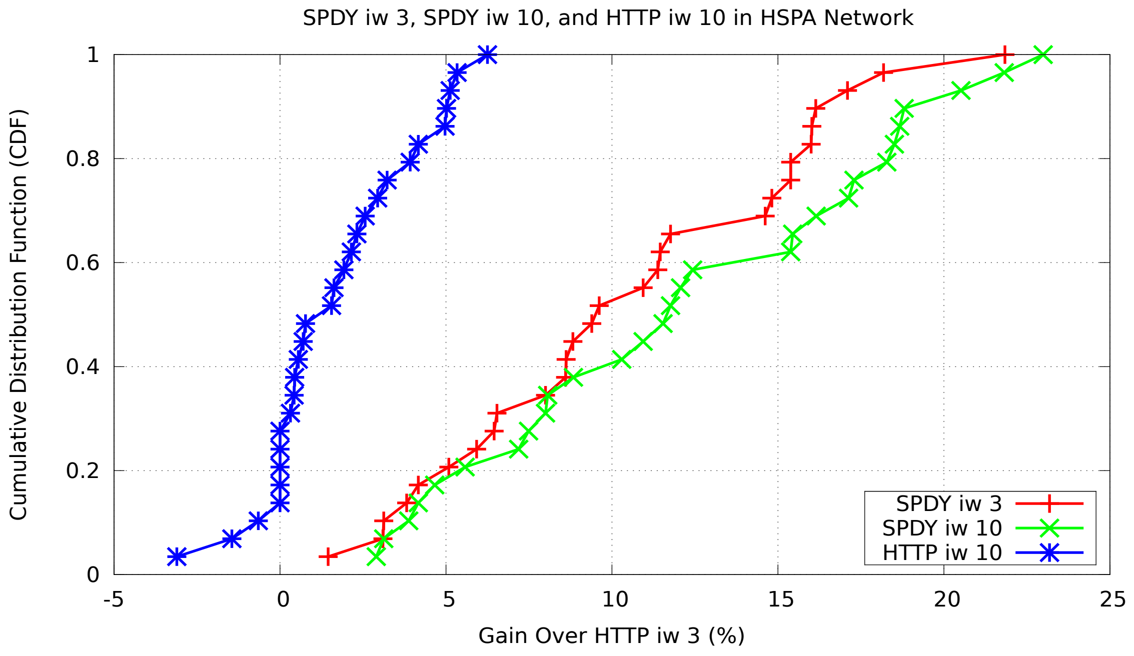

Figure 29: HSPA, Comparing HTTP and SPDY Variants

after 1.8 seconds. The value 1.8 seconds is the median of SYN to SYN-ACK delays measured in HSPA network. This is not specific to any particular website but it applies to all the websites tested over HSPA network. We removed the 1.8 seconds delay from all the page load time values and calculated the gain over HTTP with initial congestion of 3 to other protocols. The Figure 30 shows the gain over HTTP with initial congestion window of 3. We can see that if we reduce the radio delay in mobile networks, SPDY can be more efficient than HTTP.

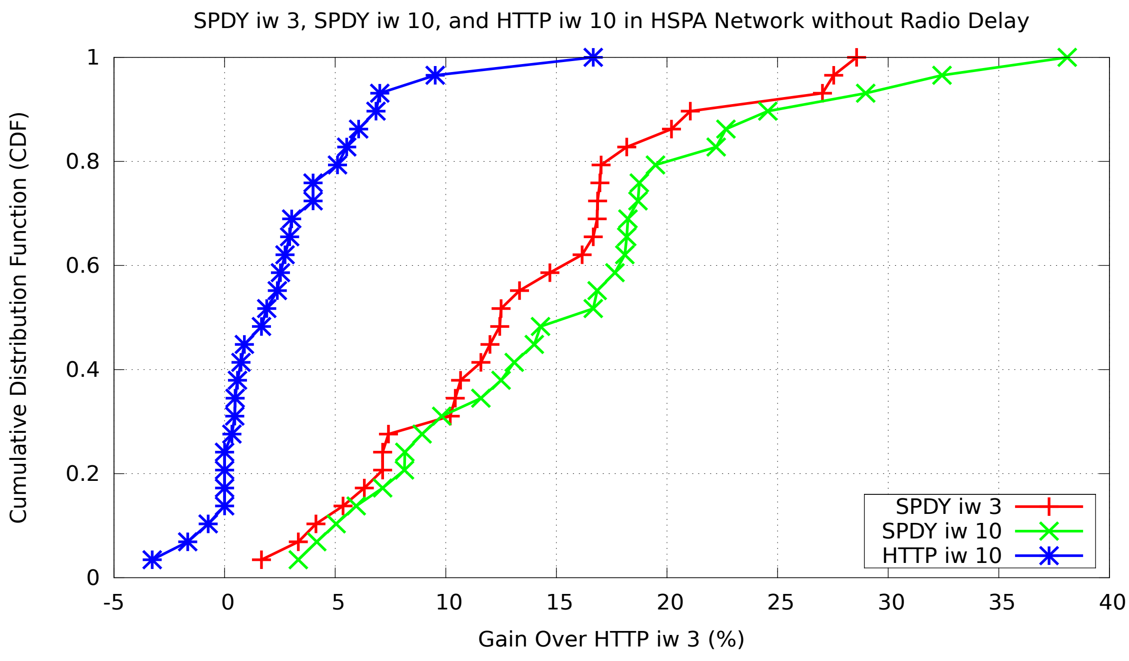

Figure 30: HSPA without Radio Delay, Comparing HTTP and SPDY Variants

5.6 Some Preliminary Tests in LTE

Some preliminary tests were conducted in LTE network of Sonera comparing SPDY and HTTP. 15 websites were tested on LTE network and some interesting conclusions were made in the process.

5.6.1 LTE Results

From Figure 31, it is very clear that in faster networks where the bandwidth is high and latency is low, the SPDY protocol is not able to make any impact in terms of performance. This is mainly due to the small RTT's. The time required to establish another TCP connection in HTTP is small hence using the less number of TCP connection of Multiplexing does not help here. Higher bandwidth offers better performance of TCP by eliminating the congestion hence there is no significant packet loss.

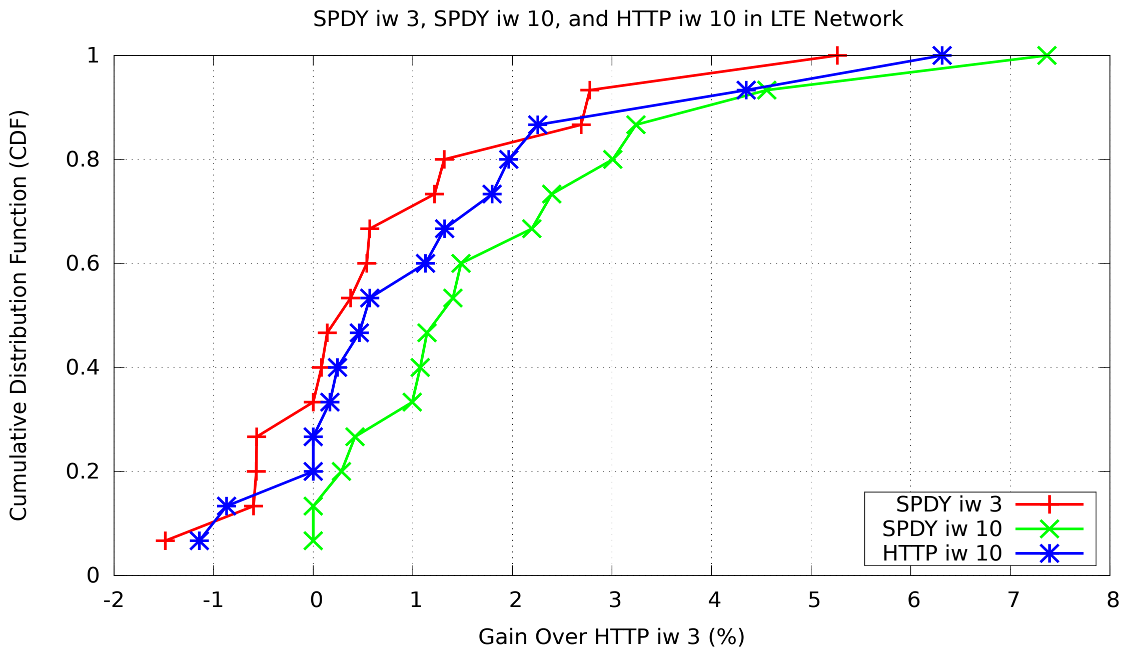

Figure 31: LTE, Comparing HTTP and SPDY Variants

There is no significant performance gain in using SPDY as the network is considered really fast in terms of bandwidth and latency. The effect of SPDY is seen only in cases with low bandwidth and high latency. The aim of this test was to figure out the effect of higher TCP initial congestion window in HTTP. RTT in LTE close to 25 ms, the extra RTT for ACK in case of initial window of three with quick-ack enabled receiver is negligible. The initial window of 10 gives a head start but initial window of 3 is optimal when the RTT is low. The case is similar with SPDY; the difference is not big with initial window of 10 or initial window of 3.

5.7 LTE Results with 200 ms Delay

To find out the behavior of SPDY in high bandwidth and high latency networks, 200 ms delay was added between the client and server machines. The value 200 ms was chosen as the most of the servers within Finland were having a RTT of 100 ms and we wanted a real world RTT value, which is comparable to the servers in North America or Asia. Hence, the delay in LTE is now equal to normal delay and the additional 200 ms. This value is estimated around 225 ms. When the latency is low, the cost of opening a new TCP connection is negligible. When the extra delay is added, a TCP connection establishment takes more than 225ms. Since SPDY is using single TCP connection per domain and HTTP used multiple TCP connection per domain, this additional delay is beneficial for SPDY protocol. Since the latency is high, BDP is also big enough to fill the wire or medium.

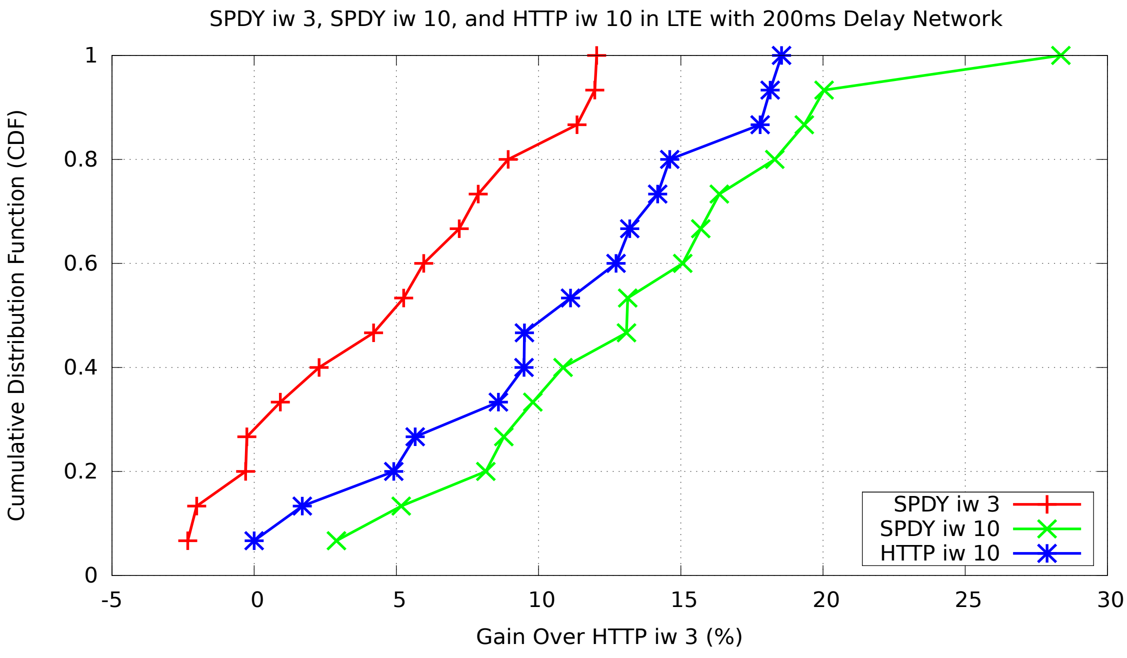

Figure 32: LTE with 200ms Delay, Comparing HTTP and SPDY variants

From Figure 32, we can clearly see the impact of SPDY in some cases and the most significant thing is the effect of initial congestion window. The TCP initial congestion window of 10 has gain over TCP initial congestion window of 3. In HSPA and GPRS network, SPDY with *initcwnd* 3 and 10 were very close in terms of performance improvement. In LTE with 200ms delay, *initcwnd* 10 in HTTP and SPDY performs better than SPDY with *initcwnd* 3.

The impact of SPDY over HTTP with added delay is seen in some cases but it is does not hold true for all websites. From a user experience point of view, the gain is not significant. The TCP initial congestion window of 10 has clear impact in all cases because the BDP is high and RTT for establishing a TCP connection is around 200 ms. This 200 ms delay can be a challenge for TCP initial window of 3 as the RTT is high and it cannot compete with TCP initial congestion window of 10. The extra delay gives added advantage of TCP initial congestion window of 10.

From the above results, it can be concluded that in LTE network with high bandwidth and low latency, the use of SPDY or HTTP do not make any significant difference in performance. The use of *initcwnd* 10 is neither beneficial nor harmful in LTE network. At the same time, high bandwidth and high latency network can benefit from SPDY as it effectively uses the multiplexing feature. The higher initial congestion window is also helpful in high latency network.

5.8 Summary of Results in All Networks

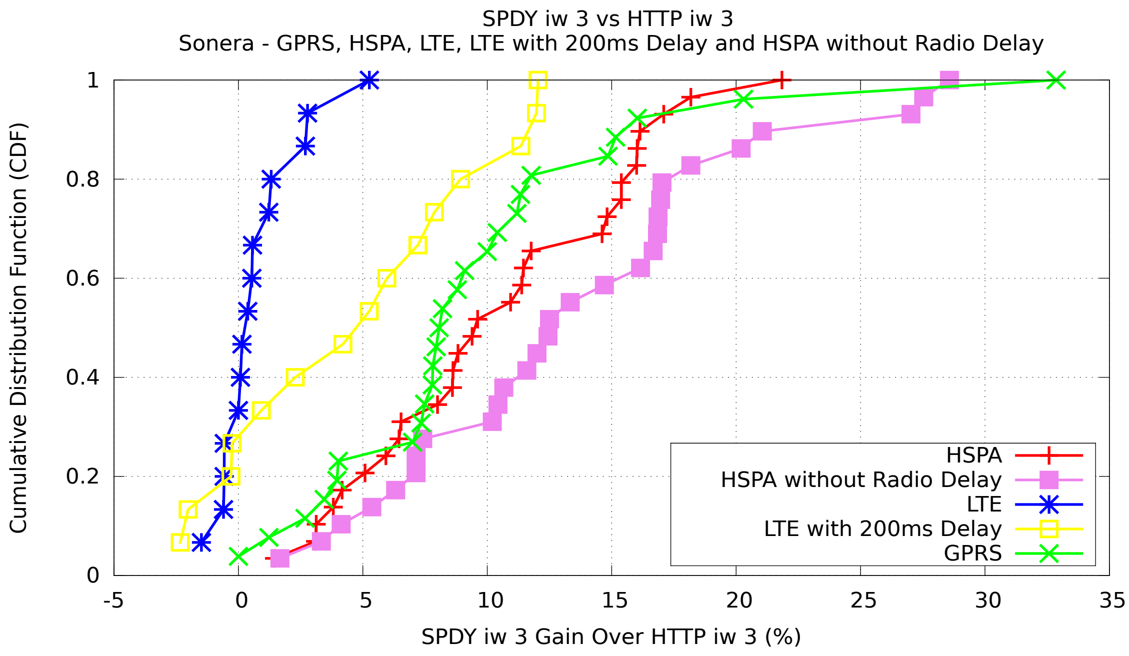

Figure 33: HTTP iw 3 vs. SPDY iw 3 in All Networks

The Figure 33 compares the SPDY with *initcwnd* 3 against HTTP with *initcwnd* 3 in all 4 types of networks. X-axis represents the SPDY *initcwnd* 3 gain over HTTP *initcwnd* 3. The gain using SPDY *initcwnd* 3 is plotted in GPRS, HSPA, HSPA without radio delay, LTE, and LTE with 200 ms delays networks.

From the Figure 33, we can see that in LTE network the use of SPDY do not give any significant gain in page load time. In LTE network with 200ms delay, we can see that depending on the website the gain can be more than 10%. One important point to keep in mind is that, in LTE networks the relative gain is low compared to GPRS or HSPA network. When we look at the GPRS network, we can see that most of the websites tested shows positive improvement in page load time. The HSPA network also exhibits the same performance as GPRS network. The HSPA without the radio delay shows advantage over SPDY as the 1.8 seconds delay is deducted from the PLT values. Hence, it can be concluded that networks with high latency and low bandwidth or networks with high latency and high bandwidth can benefit from SPDY.

The Figure 34 shows the trend of SPDY performance over different cellular access networks. The X-axis represents the RTT and the Y-axis represents the Bit rate. In GPRS network, SPDY gains around 8%, which has the highest RTT. The 8% gain is

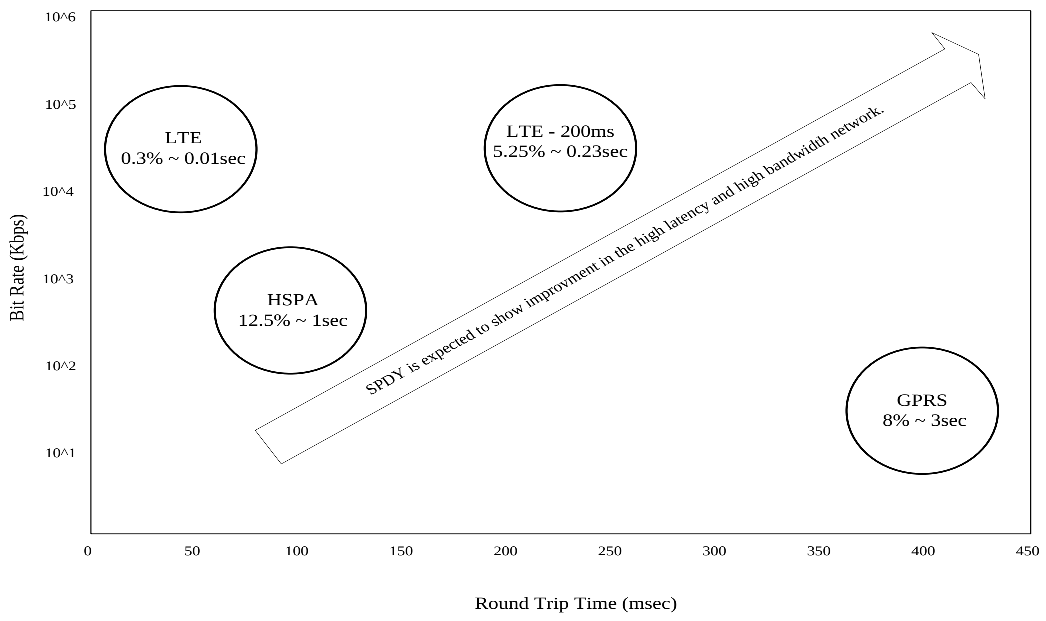

Figure 34: Comparing SPDY Performance over All Access Networks

equal to 3 seconds savings in actual PLT. The performance improvement in GPRS is mainly due to the header compression. In HSPA network without the radio delay, the gain using SPDY is 12.5%. The gain in PLT is close to 1 second in real-time. In LTE network, The SPDY has 0.3% improvement which is insignificant. When we look at the LTE network with 200ms delay, we can see that SPDY gains around 5.25%. This value translates to 0.23 seconds in real-time. All the above values are the median of PLT in different cellular access networks. Considering the trend, we can predict that SPDY will perform optimal in high bandwidth and high latency networks.

6 Related Work and Proposals

6.1 Proposals and Measurement Studies

Along with SPDY, there have been numerous attempts to improve HTTP. In the HTTP area, it is more towards enabling caching [22] [18] and adding extensions [45]. HTTP supports reverse and forward proxies [19], which makes it ideal for the addition of middle boxes. The use of DNS names to point data makes the data mobility possible with the use of CDNs. A proposal came in 2001 to replace HTTP with a protocol called Dual-HTTP (DHTTP) [49] which splits the traffic between TCP and UDP, and uses a server initiated TCP connection for transactions. A complete replacement of HTTP for another protocol is difficult because of its popularity. There are additional services like HTTP live streaming implemented on HTTP. None of these services is changing the fundamental behavior of HTTP.

From the radio perspective, there have been studies [47] [48] [11] performed on different carriers. The first study concentrates on the state promotions caused by many applications in UMTS network. The study shows how the applications can smartly use the state promotions for better energy efficiency and reduction in latency. The results and conclusion in the first study lead to the invention of Tail Optimization Protocol (TOP). TOP concentrates on the inefficient inactivity timers for the state demotions. The protocol suggests an efficient way to reduce the state demotion delays by making changes in the user-end applications. The solution has an advantage, as the changes are not implemented in cellular infrastructure. There is a recent study on the tail end power saving mechanism called TailTheft [33]. This study explores the aspect of using the tail time by effectively utilizing the resources. The study claims a save in energy consumption from 20% to 34% in different cases.

In terms of user-experience there has been many studies conducted. One of the stdies [29] conducted in Hongkong, London and Newyork in 2007 Spring, shows how the full web content and tailored web content are accepted in user groups. Depending on the geography and user community, both type of contents were accessed. One main reason for using tailored web content was the fast delivery of optimized content. When users wanted to access the full content, they preferred the full web content. The study [53], which covers the user reviews about the mobile tailored website and the hardware recommendation for the mobile devices. The paper also suggests a new generation mobile tailored website versions, which can support new protocols and formatting languages. There is another study [56], which shows the usability of mobile web browsing by comparing the mobile web browsing to desktop browsing. It mainly concentrates on the narrow long screen display and extensive scrolling issues in mobile web browsing.

On the browser development area, Amazon has introduced a browser called Silk [61] based on SPDY. The silk uses split browser architecture with the following features. The silk browser resides on mobile client and amazon cloud server. The main features of silk are shorter transit times, computing power using amazon cloud servers, persistent connection to the cloud, page indexing, and machine learning. The core networks connected to the amazon cloud ensures the short transit times. The cloud servers retrieves the web pages for the mobile browser hence the computing load and battery consumption is low. Silk always keeps a persistent connection to the amazon cloud, which reduces the time to open a new TCP connection, and incorporates multiplexing. The page-indexing feature enables the cloud server to cache the static content and make it readily available for use. The machine-learning feature is similar to server push functionality in SPDY whereas in silk the cloud server keeps the history of user navigation and learns from it.

In the TCP area, there has been also some effort like Stream Control Transmission Protocol (SCTP) [59], Structured Stream Transport (SST) [20], SMUX [21], etc. SCTP is a transport protocol mainly designed to deal with the issues with TCP such as Head-of-Line Blocking (HOL), the stream-oriented nature, the inability for multi-homing, and the DOS-like (Denial Of Service) SYN attacks. The key features of SCTP are multiplexed streams and congestion avoidance similar to TCP. There have been some comparison studies on how HTTP will behave on top of SCTP and TCP [27]. Another protocol with similar properties is the SST. The main features are the Multiplexed streams, no three-way handshakes, and Dynamic Prioritization of streams or objects. Datagrams are the main transport element of SST. There is another protocol named SMUX which is a session layer protocol providing multiplexing and stream oriented transport.

Another major initiative by Google [16] is proposed in IETF proceedings and is making some progress. The draft proposes an increase of TCP initial congestion window to 10 segments as specified in RFC 3390 [9]. The draft discusses the advantages and disadvantages of having a higher TCP initial window, the experiments conducted to prove the claims as well as some key findings about how the web applications will benefit from higher TPC initial window. The draft also discusses the cost to the network, individual connections, and the future test plans. A similar work is proposed in the draft [28] exploring an adaptive scheme for increasing the TCP initial congestion window over long time scale. The draft proposes an algorithm that is feedback based. Depending on the network errors and feedback received, the TCP initial congestion window is raised. The values, which are not suitable for the network are omitted. The adaptive scheme seems to be promising compared to the fixed value for *initcwnd*.

There is another draft [26] in IETF, which analyses the impact of higher initial congestion window. Some of the causes listed are the multiple parallel TCP connections to same domain used by browsers, badly configured buffers in the user end, and large number of embedded objects in webpages. Author also talks about the lack of support for HTTP pipelining and badly configured buffers. As a solution, the author proposes the implementation of HTTP pipelining subsequently going back to the 2 TCP connection per domain, implementation of SPDY, and different types of DIffServ deployments in user and network end.

Netalyzr [30] is a similar study done to measure properties such as NAT behavior, DNS issues, TCP/UDP service reachability, access link buffering, HTTP cache prevalence, and latency. The use of these types of tools in cellular networks can produce valuable research material.

6.2 Future Work

Features like request prioritization, server push and server hint need to be investigated further. Although Flip server replicates a commercial web server, the feasibility of implementing SPDY in production-ready servers has to be verified. Google has already deployed the SPDY protocol in their production servers for the services running over SSL. The implementation details, however, needs to be explored further. The experiments conducted in this thesis used HTTP and SPDY protocols without SSL, though the SPDY specification recommends the use the SSL. The test setup can be improved further by introducing SSL. Introducing SSL brings additional RTT's for the SSL negotiation, which might not be efficient as SPDY without SSL. There is room for exploring the SSL negotiating part and implementation details of SPDY as a part of the future work.

The SPDY needs to be tested extensively on WLAN and future networks like LTE, in order to understand its network behavior. The tests conducted for this thesis are preliminary hence there is more scope to perform some more tests in LTE with delay, packet loss, and altering the buffer size in network. The future work could also include the different network buffer size experiments comparing HTTP and SPDY. There is room for testing the SPDY on applications, which uses HTTP, E-mail services, and other application updates.

7 Conclusion

The thesis discusses the issues in mobile web browsing and network protocols such as HTTP and TCP. It investigates the current issues in mobile web browsing and proposes some ways to improve the current schemes. The proposals include radio latency reduction, DNS pre-fetching, DNS pre-resolution, HTTP pipelining, intelligent assignment of TCP connections in HTTP, and HTTP header and payload compression. We observed that some of the current browsers have already implemented or planning to implement the above mentioned improvement suggestions.

The main part of the thesis covers the alternative protocol performance in comparison to HTTP. The alternative protocol chosen for the study was SPDY protocol as it offers multiplexing, header compression, and efficient use of TCP connections. Performance results are evaluated in terms of Page Load Time of a website. The results show that websites using SPDY protocol loads the pages faster than the HTTP protocol. Although the improvement in PLT ranges from 1.44% to 21.83% over 30 websites in HSPA network and 0% to 32.85% in 26 websites on GPRS network. It can be more or less when we use real world web servers instead of our customized server for testing. One of the mains facts that came out in the experiments is the importance of header compression. The header compression and reduction in redundant information in headers plays a major role in reducing the payload on the wire or medium. In addition, multiplexing not only saves the time to establish a new TCP connection but also benefits from the higher TCP window in established TCP connections.

One of the important lessons is the handling of TCP connections in HTTP. The main advantage of opening multiple TCP connections in the beginning of the web transaction with higher initial congestion window leads to the avoidance slowstart of newly established TCP connections. It also avoids the TCP connection establishment delay of one RTT. Websites should limit the number of objects, as less number of objects leads to less number of TCP connections. If the existing TCP connections are reused intelligently by calculating the highest TCP window size, then the rate of transfer will be higher. Multiplexing can be helpful to send multiple requests and receive multiple response chunks using same connection. HTTP pipelining suffers from HOL and responses can be blocked for a long time. One way to circumvent this problem is to issue the pipelining requests intelligently, where the browser has the knowledge of blocking and non-blocking requests.

Preliminary tests in LTE clearly shows that there is no significant gain in using SPDY when there is high bandwidth and low latency. The importance of SPDY is only seen when the network bandwidth is high and latency is also high. In these kinds of networks, the time for establishing a TCP connection high during the web transaction and it makes the web transaction long. In LTE tests with 200 ms delay, it was observed that TCP initial congestion window of 10 is beneficial.

A complete change of protocols on server and client side is not a cost efficient method. Implementing SPDY in current network require considerable changes on the server side and additional plug-ins on the browser side. SPDY mandates the use of TLS, which may not be very convincing for popular websites present today. Another proposal can be made avoiding the track of SPDY (or any other new protocol implementation) and try to solve the performance issues by i) reducing the radio transition delays, ii) DNS pre-fetching, iii) DNS pre-resolution, iv) HTTP payload and header compression, v) TLS compression/improvement, vi) Websocket, vii) efficient website structure, and viii) intelligent use of TCP connections. The future cannot be predicted easily but we can observe, propose, work, and wait to see what will happen to the mobile web browsing.

References

[1] 3GPP. GPRS and EDGE. `http://www.3gpp.org/article/gprs-edge`, December 2011.

[2] 3GPP. HSPA. `http://www.3gpp.org/HSPA`, December 2011.

[3] A. Bittau and D. Boneh and M. Hamburg and M. Handley and D. Mazieres and Q. Slack. Cryptographic protection of TCP Streams. Internet-draft, March 2011. Work in progress.

[4] A. Jain and J. Glasgow. Use compression to make the web faster. `http://code.google.com/speed/articles/use-compression.html`, May 2010.

[5] A. Langley. Transport Layer Security (TLS) Snap Start. Internet-draft, June 2010. Work in progress.

[6] A. Langley and N. Modadugu and B. Moeller. Transport Layer Security (TLS) False Start. Internet-draft, June 2010. Work in progress.

[7] M. Allman, K. Avrachenkov, U. Ayesta, J. Blanton, and P. Hurtig. Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP). RFC 5827 (Experimental), May 2010.

[8] M. Allman, H. Balakrishnan, and S. Floyd. Enhancing TCP's Loss Recovery Using Limited Transmit. RFC 3042 (Proposed Standard), January 2001.

[9] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. RFC 3390 (Proposed Standard), October 2002.

[10] B. Lawson. Opera Presto 2.1 - Web standards supported by Opera core. `http://dev.opera.com/articles/view/presto-2-1-web-standards-supported-by/`, September 2008.

[11] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 280–293, New York, NY, USA, 2009. ACM.

[12] Chromium Project. The Chromium repository Website. `http://build.chromium.org/buildbot/continuous/`, June 2011.

[13] Cisco. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2010-2015. `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html`, Feb 2011.

[14] D. Wing and A. Yourtchenko. Happy Eyeballs: Trending Towards Success with Dual-Stack Hosts. Internet-draft, May 2011. Work in progress.

[15] R. Draves. Default Address Selection for Internet Protocol version 6 (IPv6). RFC 3484 (Proposed Standard), February 2003.

[16] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An argument for increasing tcp's initial congestion window. *SIGCOMM Comput. Commun. Rev.*, 40:26–33, June 2010.

[17] S. Parkvall E. Dahlman and J. Skold. *4G: LTE/LTE-Advanced for Mobile Broadband*. Academic Press, 2011.

[18] J. Erman, A. Gerber, M. Hajiaghayi, P. Dan, S. Sen, and O. Spatscheck. To cache or not to cache: The 3g case. *Internet Computing, IEEE*, 15(2):27 –34, march-april 2011.

[19] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266.

[20] B. Ford. Structured streams: a new transport abstraction. *SIGCOMM Comput. Commun. Rev.*, 37:361–372, August 2007.

[21] J. Gettys and H. F. Nielsen. SMUX protocol specification. W3C working draft, W3C, July 1998. http://www.w3.org/TR/1998/WD-mux-19980710.

[22] H. Hassanein, Zhengang Liang, and P. Martin. Performance comparison of alternative web caching techniques. In *Computers and Communications, 2002. Proceedings. ISCC 2002. Seventh International Symposium on*, pages 213 – 218, 2002.

[23] D. Hyatt and I. Hickson. HTML 5. W3C working draft, W3C, August 2009. http://www.w3.org/TR/2009/WD-html5-20090825/.

[24] Issue 8991. Pipelining in Google Chrome. `http://code.google.com/p/chromium/issues/detail?id=8991`, March 2009.

[25] J. Chu and N. Dukkipati and Y. Cheng and M. Mathis. Increasing TCP's Initial Window. Internet-draft, January 2011. Work in progress.

[26] J. Gettys. IW10 Considered Harmful. Internet-draft, August 2011. Work in progress.

[27] J. T. Leighton. Comparison of HTTP over TCP and SCTP in High Delay Networks. `http://www.cis.udel.edu/~leighton/firefox.html`, July 2011.

[28] J. Touch. Automating the Initial Window in TCP draft-touch-tcpm-automatic-iw-00.txt. `http://tools.ietf.org/html/draft-touch-tcpm-automatic-iw-00`, December 2010. Work in progress.

[29] A. Kaikkonen. Full or tailored mobile web- where and how do people browse on their mobiles? In *Proceedings of the International Conference on Mobile Technology, Applications, and Systems*, Mobility '08, pages 28:1–28:8, New York, NY, USA, 2008. ACM.

[30] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: illuminating the edge network. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 246–259, New York, NY, USA, 2010. ACM.

[31] K. Lagar-Cavilla, H. A.and Joshi, A. Varshavsky, J. Bickford, and D. Parra. Traffic backfilling: subsidizing lunch for delay-tolerant applications in umts networks. *SIGOPS Oper. Syst. Rev.*, 45(3):77–81, January 2012.

[32] Y. Lin, Y. Haung, Y. Chen, and I. Chlamtac. Mobility management: from gprs to umts. *Wireless Communications and Mobile Computing*, 1(4):339–359, 2001.

[33] H. Liu, Y. Zhang, and Y. Zhou. Tailtheft: leveraging the wasted time for saving energy in cellular communications. In *Proceedings of the sixth international workshop on MobiArch*, MobiArch '11, pages 31–36, New York, NY, USA, 2011. ACM.

[34] M. Belshe and R. Peon. SPDY Protocol Draft-2. `https://sites.google.com/a/chromium.org/dev/spdy/spdy-protocol/spdy-protocol-draft2`, July 2011.

[35] M. Belshe and R. Peon. SPDY:An experimental protocol for faster web. `https://sites.google.com/a/chromium.org/dev/spdy/spdy-whitepaper`, July 2011.

[36] M. Crowley. *Pro Internet Explorer 8 and 9 Development: Developing Powerful Applications for the Next Generation of IE.* Apress, April 2010.

[37] M. Stachowia. Pipelining in Safari. `http://www.webkit.org/blog/75/optimizing-page-load-time-and-a-little-about-the-debug-menu`, October 2006.

[38] Mac OS Forge. The WebKit Open Source Project. `http://www.webkit.org/`, August 2011.

[39] Mozilla Developer Network. Gecko. `https://developer.mozilla.org/en/gecko`, July 2011.

[40] MozillaZine. Pipelining in Mozilla Firefox. `http://kb.mozillazine.org/Network.http.pipelining`, September 2010.

[41] msdn.microsoft.com. TCP Receive Window Size and Window Scaling in Windows OS. `http://msdn.microsoft.com/en-us/library/ms819736.aspx`, July 2011.

[42] Opera.com. Pipelining in Opera. `http://www.opera.com/press/releases/2000/03/28/`, March 2000.

[43] Opera.com. Opera's Settings File Explained. `http://www.opera.com/support/usingopera/operaini/index.dml#performance`, July 2011.

[44] C. Pluntke, L. Eggert, and N. Kiukkonen. Saving mobile device energy with multipath tcp. In *Proceedings of the sixth international workshop on MobiArch*, MobiArch '11, pages 1–6, New York, NY, USA, 2011. ACM.

[45] L. Popa, A. Ghodsi, and I. Stoica. Http as the narrow waist of the future internet. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets '10, pages 6:1–6:6, New York, NY, USA, 2010. ACM.

[46] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Profiling resource usage for mobile applications: a cross-layer approach. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 321–334, New York, NY, USA, 2011. ACM.

[47] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3g networks. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 137–150, New York, NY, USA, 2010. ACM.

[48] F. Qian, Z. Wang, A. Gerber, Z.M. Mao, S. Sen, and O. Spatscheck. Top: Tail optimization protocol for cellular radio resource allocation. In *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, pages 285 – 294, oct. 2010.

[49] M. Rabinovich and H. Wang. Dhttp: an efficient and cache-friendly transfer protocol for web traffic. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1597 –1606 vol.3, 2001.

[50] V. Ramasubramanian and E. Sirer. The design and implementation of a next generation name service for the internet. *SIGCOMM Comput. Commun. Rev.*, 34:331–342, August 2004.

[51] S. Ramachandran. Web metrics: Size and number of resources. `http://code.google.com/speed/articles/web-metrics.html`, May 2010.

[52] S. Souders. Sharding Dominant Domains. `http://www.stevesouders.com/blog/2009/05/12/sharding-dominant-domains/`, May 2009.

[53] G. Schmiedl, M. Seidl, and K. Temper. Mobile phone web browsing: a study on usage and usability of the mobile web. In *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '09, pages 70:1–70:2, New York, NY, USA, 2009. ACM.

[54] J.and Mathis M. Semke, J.and Mahdavi. Automatic tcp buffer tuning. *SIGCOMM Comput. Commun. Rev.*, 28:315–323, October 1998.

[55] S. Shrestha. Mobile web browsing: usability study. In *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, Mobility '07, pages 187–194, New York, NY, USA, 2007. ACM.

[56] S. Shrestha. Mobile web browsing: usability study. In *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, Mobility '07, pages 187–194, New York, NY, USA, 2007. ACM.

[57] squid-cache.org. Pipelining support in Squid. `http://www.squid-cache.org/Doc/config/pipeline_prefetch/`, July 2011.

[58] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001 (Proposed Standard), January 1997. Obsoleted by RFC 2581.

[59] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol. RFC 2960 (Proposed Standard), October 2000. Obsoleted by RFC 4960, updated by RFC 3309.

[60] T. Savolainen. Experience of Host Behavior in Broken IPv6 Networks. `http://www.ietf.org/proceedings/80/slides/v6ops-12.pdf`, March 2011.

[61] The Amazon Silk Team. Amazon Silk. `http://amazonsilk.wordpress.com/`, September 2011.

[62] L. Wang, A. Ukhanova, and E. Belyaev. Power consumption analysis of constant bit rate data transmission over 3g mobile wireless networks. In *ITS Telecommunications (ITST), 2011 11th International Conference on*, pages 217 –223, aug. 2011.

Appendix

A Results of Other Websites Tested

The numerical value of page load time for different websites are listed below. The table gives information on website, protocol used, PLT, and percentage gain in PLT compared to HTTP with initial congestion window of 3.

Websites and PLT in Sec(Median)				
Megatuutti.fi	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	2.4	2.3	2.3	2.3
Gain (%)	-	4.2%	4.2%	4.2%
PLT in GPRS (s)	3.3	3.3	3.0	3.1
Gain (%)	-	0%	9.1%	6.1%
Baidu.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	3.2	3.2	3.1	3.1
Gain (%)	-	0%	3.1%	3.1%
PLT in GPRS (s)	7.0	7.0	4.7	4.8
Gain (%)	-	0%	32.9%	31.4%
Kernel.org	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	3.9	3.7	3.3	3.1
Gain (%)	-	5.1%	15.4%	20.5%
PLT in GPRS (s)	19	19	14	14
Gain (%)	-	0%	26.3%	26.3%
Wikipedia.org	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	5.0	5.0	4.6	4.6
Gain (%)	-	0%	8%	8%
PLT in GPRS (s)	24	24	21	21
Gain (%)	-	0%	12.5%	12.5%
Craiglist.org	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	5.1	4.9	4.5	4.5
Gain (%)	-	3.9%	11.8%	11.8%
PLT in GPRS (s)	17	16	15	14
Gain (%)	-	5.9%	11.8%	17.6%
Facebook.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	5.2	5.1	4.7	4.6
Gain (%)	-	1.9%	9.6%	11.5%
PLT in GPRS (s)	33	32	30	31
Gain (%)	-	3.0%	9.1%	6.1%

Table 8: Websites and PLT Table 1

Websites and PLT in Sec(Median)				
Bing.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	5.5	5.4	4.5	4.3
Gain (%)	-	1.8%	18.2%	21.8%
PLT in GPRS (s)	27	27	23	23
Gain (%)	-	0%	14.8%	14.8%
Linkedin.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	6.8	6.6	6.2	6.1
Gain (%)	-	2.9%	8.8%	10.3%
PLT in GPRS (s)	34	33	30	30
Gain (%)	-	2.9%	11.8%	11.8%
Youtube.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	7.5	7.1	6.3	6.1
Gain (%)	-	5.3%	16.0%	18.7%
PLT in GPRS (s)	49	49	47	47
Gain (%)	-	0%	4.1%	4.1%
Wordpress.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	8.7	8.7	6.8	6.7
Gain (%)	-	0%	21.8%	23%
PLT in GPRS (s)	83	83	76	77
Gain (%)	-	0%	8.4%	7.2%
Ebay.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	9.3	9.0	8.5	7.6
Gain (%)	-	3.2%	8.6%	18.3%
PLT in GPRS (s)	64	64	58	57
Gain (%)	-	0%	9.4%	10.9%
Nokia.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	10	10	6.5	6.4
Gain (%)	-	0%	35%	36%
PLT in GPRS (s)	31	31	29	28
Gain (%)	-	0%	6.5%	9.7%
Yahoo.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	11	11	9.6	9.5
Gain (%)	-	0%	12.7%	13.6%
PLT in GPRS (s)	89	83	72	72
Gain (%)	-	6.7%	19.1%	19.1%
Amazon.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	11	11	10	10
Gain (%)	-	0%	9.1%	9.1%
PLT in GPRS (s)	131	128	110	111
Gain (%)	-	2.3%	16.0%	15.3%

Table 9: Websites and PLT Table 2

Websites and PLT in Sec(Median)				
Imdb.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	12	12	11	10
Gain (%)	-	0%	8.3%	16.7%
PLT in GPRS (s)	-	-	-	-
Gain (%)	-	-	-	-
Ovi.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	12	12	10	10
Gain (%)	-	0%	16.7%	16.7%
PLT in GPRS (s)	112	114	109	109
Gain (%)	-	-1.8%	2.7%	2.7%
Ask.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	13	13	13	13
Gain (%)	-	0%	0%	0%
PLT in GPRS (s)	61	64	59	59
Gain (%)	-	-4.9%	3.3%	3.3%
Nordea.fi	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	14	14	8.6	8.7
Gain (%)	-	0%	38.6%	37.9%
PLT in GPRS (s)	18	18	18	17
Gain (%)	-	0%	0%	5.6%
Spotify.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	15	16	14	14
Gain (%)	-	-6.7%	6.7%	6.7%
PLT in GPRS (s)	162	165	162	161
Gain (%)	-	-1.9%	0%	0.6%
Yle.fi	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	18	18	17	17
Gain (%)	-	0%	5.6%	5.6%
PLT in GPRS (s)	163	162	161	161
Gain (%)	-	0.6%	1.2%	1.2%
Bbc.co.uk	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	21	21	18	17
Gain (%)	-	0%	14.3%	19%
PLT in GPRS (s)	134	137	119	118
Gain (%)	-	-2.2%	11.2%	11.9%
Aol.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	18	17	16	16
Gain (%)	-	5.6%	11.1%	11.1%
PLT in GPRS (s)	150	144	133	141
Gain (%)	-	4.0%	11.3%	6.0%

Table 10: Websites and PLT Table 3

Websites and PLT in Sec(Median)				
Cnn.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	20	19	17	17
Gain (%)	-	5%	15%	15%
PLT in GPRS (s)	200	201	186	184
Gain (%)	-	-0.5%	7.0%	8.0%
Tumblr.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	22	22	20	21
Gain (%)	-	0%	9.1%	4.5%
PLT in GPRS (s)	-	-	-	-
Gain (%)	-	-	-	-
Qq.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	23	23	22	20
Gain (%)	-	0%	4.3%	13%
PLT in GPRS (s)	80	82	66	75
Gain (%)	-	-2.5%	17.5%	6.3%
Espn.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	25	25	24	24
Gain (%)	-	0%	4%	4%
PLT in GPRS (s)	173	171	155	156
Gain (%)	-	1.2%	10.4%	9.8%
mtv3.fi	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	29	30	27	28
Gain (%)	-	-3.4%	6.9%	3.4%
PLT in GPRS (s)	-	-	-	-
Gain (%)	-	-	-	-
Iltalehti.fi	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	32	31	28	28
Gain (%)	-	3.1%	12.5%	12.5%
PLT in GPRS (s)	-	-	-	-
Gain (%)	-	-	-	-
Cnet.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	33	32	23	23
Gain (%)	-	3%	30.3%	30.3%
PLT in GPRS (s)	202	201	161	160
Gain (%)	-	0.5%	20.3%	20.8%
Nytimes.com	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	34	33	29	28
Gain (%)	-	2.9%	14.7%	17.6%
PLT in GPRS (s)	202	203	194	194
Gain (%)	-	-0.5%	4%	4%

Table 11: Websites and PLT Table 4

Websites and PLT in Sec(Median)				
Hs.fi	HTTP - iw03	HTTP - iw10	SPDY - iw03	SPDY - iw10
PLT in HSPA (s)	35	36	33	30
Gain (%)	-	-2.9%	5.7%	14.3%
PLT in GPRS (s)	-	-	-	-
Gain (%)	-	-	-	-

Table 12: Websites and PLT Table 5