ABSTRACT

Title of Thesis: INTERNET CONTENT DELIVERY ACCELERATION METHODS FOR HYBRID NETWORK TOPOLOGIES

Aniruddha Ramesh Bhalekar, Master of Science, 2003

Thesis directed by: Professor John S. Baras
Department of Electrical and Computer Engineering

Internet demand has been characterized by an exponential growth. Broadband Internet via satellite provides high-speed direct-to-home (DTH) Internet access. However, such systems suffer from high delay which causes degradation in the performance of protocols such as HTTP and TCP, which were developed to be used on terrestrial networks, in terms of throughput and the latency perceived by the end-user of the system.

In this thesis we investigate different versions and developments of the HTTP protocol and different algorithms and schemes in order to reduce the user's perceived latency and make browsing DTH delivered Internet a less cumbersome experience. The

first of the two suggested schemes uses concurrent connections and the second scheme is a caching scheme. We show that there is a marked reduction in the perceived delay using these methods. Lastly, this thesis describes other non-mature mechanisms and their possible implementation to overcome the above-mentioned problem.

INTERNET CONTENT DELIVERY ACCELERATION METHODS FOR HYBRID NETWORK TOPOLOGIES

by

Aniruddha Ramesh Bhalekar

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2003

Advisory Committee:

Professor John S. Baras, Chair
Professor Virgil Gligor
Dr. Michael Hadjitheodosiou

DEDICATION

To,

Aai, Baba, Tai and Arundhati

ACKNOWLEDGEMENTS

I am grateful to my advisor Dr. John S. Baras, for his support, encouragement and guidance throughout my time at the university. This work would have not been possible without him. I sincerely appreciate the time and helpful comments given my Dr. Virgil Gligor and Dr. Michael Hadjitheodosiou, who agreed to serve on my thesis committee.

I would also like to thank Doug Dillon from HNS for his time and interest, Althia Kirlew for her invaluable help, Manish Karir for his advice and insights, and Diane Hicks for everything at the CSHCN. I would also like to express my appreciation for the financial support granted by Hughes Network Systems (HNS) and the National Science Foundation (NSF) under co-operative agreements 2612 and 3027 under the Maryland Industrial Partnerships (MIPS) program. I am also grateful to Intelsat Global Services Corporation, Washington DC for the summer internship opportunity from which I learnt a lot.

I am indebted to my friends, mentors and colleagues for helping me complete this work, to my parents for their constant encouragement and finally to Arundhati for her infinite support and patience.

TABLE OF CONTENTS

1 Introduction

This thesis presents a performance evaluation of Internet transfers and specifically World Wide Web (WWW) page retrievals over networks which have a high-delay segment such as a satellite link. In such hybrid network topologies, protocols that were traditionally built to be used over terrestrial-only network topologies are used. For instance, the Hyper Text Transfer Protocol (HTTP) is often used as the application layer protocol and the Transmission Control Protocol (TCP) is used as the transport layer protocol over network paths which include a satellite component.

As an overview, we first discuss the nature of the HTTP protocol and evaluate its performance over satellite links. We define the problem we are trying to alleviate and focus on the metrics we consider. We then present a comparison of the performance of the different versions of HTTP and its various developments. We analyze the effect of these developments on the performance of the protocol in terms of the metrics we have considered.

We then briefly explain the motivation for our proposed schemes. We describe the optimized browser scheme and why we feel that it will contribute to reduce user-perceived latency. At this point we look at the previous work on this concept and realize how this solution performs differently on the network under consideration as compared to its performance on links of traditional networks which are characterized by intermediate routers and low delays. We go on to briefly explain the implementation of the testbed and detail the different scenarios tested on it. We then compare the results of the experiments

run and prove that this scheme does indeed have a better performance than others mentioned before.

We then describe cumulative caching and explain the concept in detail. We discuss the benefits of this scheme on, not only the user-perceived latency, but also on the bandwidth utilization on the space segment of the link to the Network Operations Center (NOC). While discussing the concept of the scheme we present the algorithm that needs to be implemented to make this simple but efficient scheme work. Later, via analytical arguments we qualify the impact of this scheme on the perceived latency by the end user while browsing the Internet. We conclude the major part of this thesis by stating out conclusions and the possible future work that can be done in this area at this point.

We then discuss the fast delta scheme, the pre-creating of deltas and the pre-fetching of web resources briefly. We discuss the motivation for these schemes, the underlying algorithms, the possible implementation of these schemes and associated issues.

1.1 Overview and Motivation

HTTP is a request-response type application layer protocol. In our case the transport layer protocol is TCP Reno. This is because, currently, Reno is the most popular and widely used version of TCP for the WWW. TCP Reno incorporates Slow Start (SS), Congestion Avoidance, Fast Retransmit and the Fast recovery algorithms [1]. A HTTP transaction between a client and server involves setting up a HTTP over TCP connection between the two. This involves the traditional 3-way TCP handshake for connection

establishment. The network we consider is the Hughes Network Systems SPACEWAY®. This is a two-way DTH broadband Internet service. Two-way implies that the incoming data to the end user's terminal (client) from the Internet and the outgoing data from the client to the web-server use the satellite segment. The basic network topology is as shown in Figure 1.1.

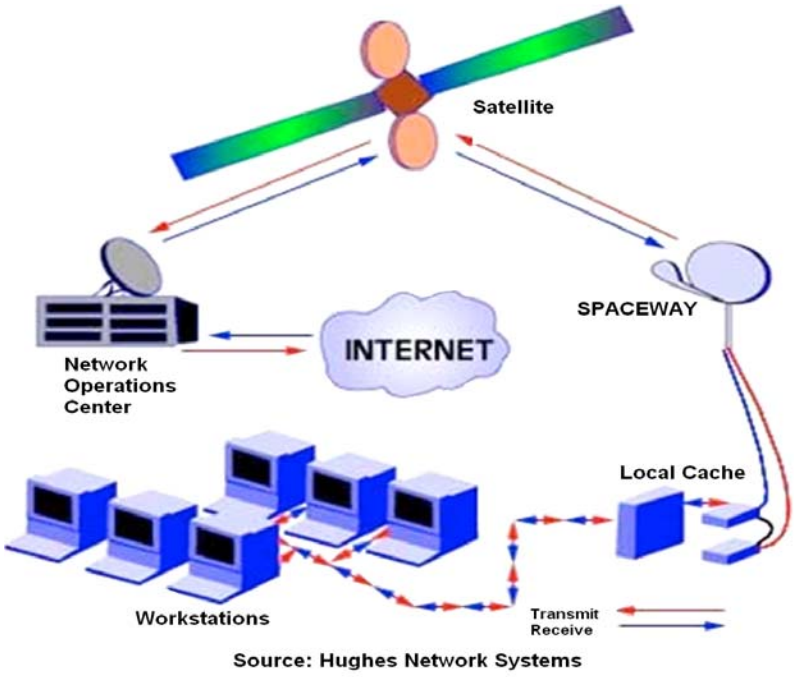

Figure 1.1: Network under consideration

The satellite in consideration is a Geo-synchronous Earth Orbit (GEO), PANAMSAT Galaxy XI Satellite. Due to the physical distance between the satellite and earth stations (22,282 miles) the time it takes for radio waves to reach the satellite, once transmitted by the earth station is more than $1/8^{th}$ of a second. The transit time on the downlink from the satellite to the hub (NOC) is also greater than $1/8^{th}$ of a second. Hence the Round-Trip Time (RTT) i.e. from the remote earth station to the satellite to the hub

and from the hub to the satellite and back to the earth station can be larger than half a second.

This delay amplifies the effect of the TCP triple handshake which is required for a connection to be set up between a client and a server. Also, in TCP, the Maximum Segment Size (MSS) is typically around 536 bytes. The HTTP requests of the client may sometimes be larger than the TCP segment size and hence span multiple RTTs. This overhead is in addition to the initial connection setup overhead of HTTP over TCP to get through to the server [2]. TCP's Slow Start algorithm aggravates this delay as the second packet cannot be sent until the first has been acknowledged. Hence the HTTP response may require multiple round trips to reach the server in the first place. The server responds to this request, after it has fully arrived, by sending the requested web resource to the client. This web resource too, may take multiple RTTs to reach the client. This causes significantly increased user-perceived latency due to HTTP transactions on this segment. The latency becomes apparent to the end-user when the transaction between the client and the server is actually a request from a personal computer to a web server and the response information is the base HTML web-page along with objects embedded in the base page. This makes "web-browsing" a cumbersome experience.

The reduction of this delay, which is perceived by the end user on the client machine, is the very issue that we try to address and resolve in this thesis. We focus only on developments at the application layer such as those that deal with HTTP enhancements, caching and other related schemes. TCP as a transport layer protocol has its share of problems when dealing with transfers on satellite links such as the default assumption that all losses in a network are congestion losses, the Slow Start algorithm,

the accumulation of TIME_WAIT states [2], etc. These issues are not addressed in this thesis. We suggest schemes to reduce the user's perceived latency using methods that operate only on the application layer.

1.2 Thesis Organization

In this thesis we examine the issues related to HTTP over satellite links. We propose solutions to the increased user-perceived latency problem in such networks. We demonstrate the implementation of these schemes and perform experiments to explore their feasibility and practicality.

The remainder of this thesis is organized as follows. In chapter 2 we discuss various HTTP versions and developments. We look at the performance of FTP, HTTP/1.0 and HTTP/1.1. We also look at other HTTP related developments such as persistent HTTP (p-HTTP) and the usage of conditional statements, which try to alleviate this problem. Chapter 3 provides a discussion of other work in this area, which contributes towards mitigating the effects of perceived latency. We look at non-HTTP schemes like data compression, delta encoding to be used along with cache validation. A scheme that deals with changing the basic nature of web content is discussed. We also look at some TCP issues and mention schemes that proposed using another transport layer protocol and a scheme that experimented with the use of a larger initial TCP cwnd. The pros and cons of all of the schemes mentioned above along with the pros and cons of HTTP state management schemes are discussed in this chapter. Chapters 2 and 3 hence give us a very good idea of the work done so far and the work currently underway in this area.

Chapter 4 discusses the first proposed scheme, known as the optimized browser scheme, to solve the above-mentioned problem. In this chapter we follow the logical sequence of the motivation of the scheme in terms of adaptive connections. For the sake of continuity, we have tried to include the entire scheme from its motivation to its evolution to the final version and conclusion in the detailed sections of this one chapter. Section 4.2 describes the implementation of the scheme and gives the details of the implementation of this scheme, in terms of its simulation setup and test scenarios that were developed. Sections 4.3 and 4.4 discuss the experiments run and the observations and results in terms of the performance of the tested schemes and the interpretations of those results, respectively. We then state the conclusions of these experiments in section 4.5. The next section is a critical evaluation of the scheme and describes the issues that exist with a scheme such as the adaptive connections scheme. We then show the logical maturing and evolution of the scheme from adaptive connections to the optimized browser scheme and discuss the pros and cons of the scheme in section 4.7. This section is followed by the summary and conclusions of the scheme and chapter in section 4.8.

Chapter 5 discusses the cumulative cache scheme in its entirety. The approach used to explain the scheme here is the same as is used for the Optimized Browser scheme in the previous chapter. Sections 5.1 and 5.2 introduce the concept of a cumulative cache and provide the necessary background for this scheme, respectively. The next section explains the motivation for this scheme, targeting the specific network topology in which it will be successful and also explains the algorithm in detail. The two sub-sections of the next section, i.e. 5.4 give the related work in this area and hence lay the foundation for the quantification of the benefits through a theoretical approach. Section 5.5 then states

the observations and details the benefits that can be obtained by the implementation of the cumulative cache scheme. Section 5.6 touches upon the problems that need to be considered when the scheme is implemented and the issues involved in this area. Section 5.7 concludes this chapter and states possible future work in this area.

Chapter 6 provides the conclusions of the major portion of this thesis. It deals with summarizing the schemes discussed in the previous two chapters and states the pros and cons of each scheme.

Chapter 7, which is the last chapter in this thesis, describes future work specific to the optimized browser scheme and the cumulative cache scheme discussed here. The latter part of the chapter also includes a discussion of two other solutions to the perceived latency problem that have been explored by us. The first of these other schemes is known as the fast delta algorithm and the other involves the pre-creating of deltas and pre-fetching of web resources. Both the schemes are not yet fully mature. They have been described in the thesis in order to provide the reader with the full picture about past and current work in this area, along with a head start for future work.

2 HTTP Developments

In this chapter we take a look at the different versions of HTTP and note how the protocol has evolved. We compare HTTP's most popular versions i.e. versions 1.0 and 1.1. Note that HTTP versions will now onwards referred to as HTTP/x.x throughout this thesis, where "x.x" denotes the version number. For instance, HTTP version 0.9 will be referred to as HTTP/0.9. We also look at the steps taken to modify the HTTP protocol to reduce the impact of the delay in the link on the actual performance, not through a newer version, but via developments or upgrades.

2.1 HTTP/1.0

HTTP/1.0 was developed as a better performing alternative as compared to its predecessor HTTP/0.9 and File Transfer Protocol (FTP). The primary differences between HTTP/0.9 and HTTP/1.0 were that, using the latter, the server could now identify the version number of the protocol because it was included in the request line; both the request and response could now contain variable number of header fields in HTTP/1.0 and the first line of the HTTP response now included a 3-digit status code along with the version number of the protocol [3].

2.1.1 Advantages Using HTTP/1.0

As compared to FTP, HTTP/1.0 required one less RTT (Round Trip Time) in the first transaction. HTTP/1.0 also required one less RTT as compared to FTP for all subsequent transactions. The first FTP transaction has three RTT's of overhead before the

file moves from the server to the client. Subsequent requests are fulfilled after two RTT's of overhead. This becomes clear diagrammatically. Refer to figure 1.2 below.

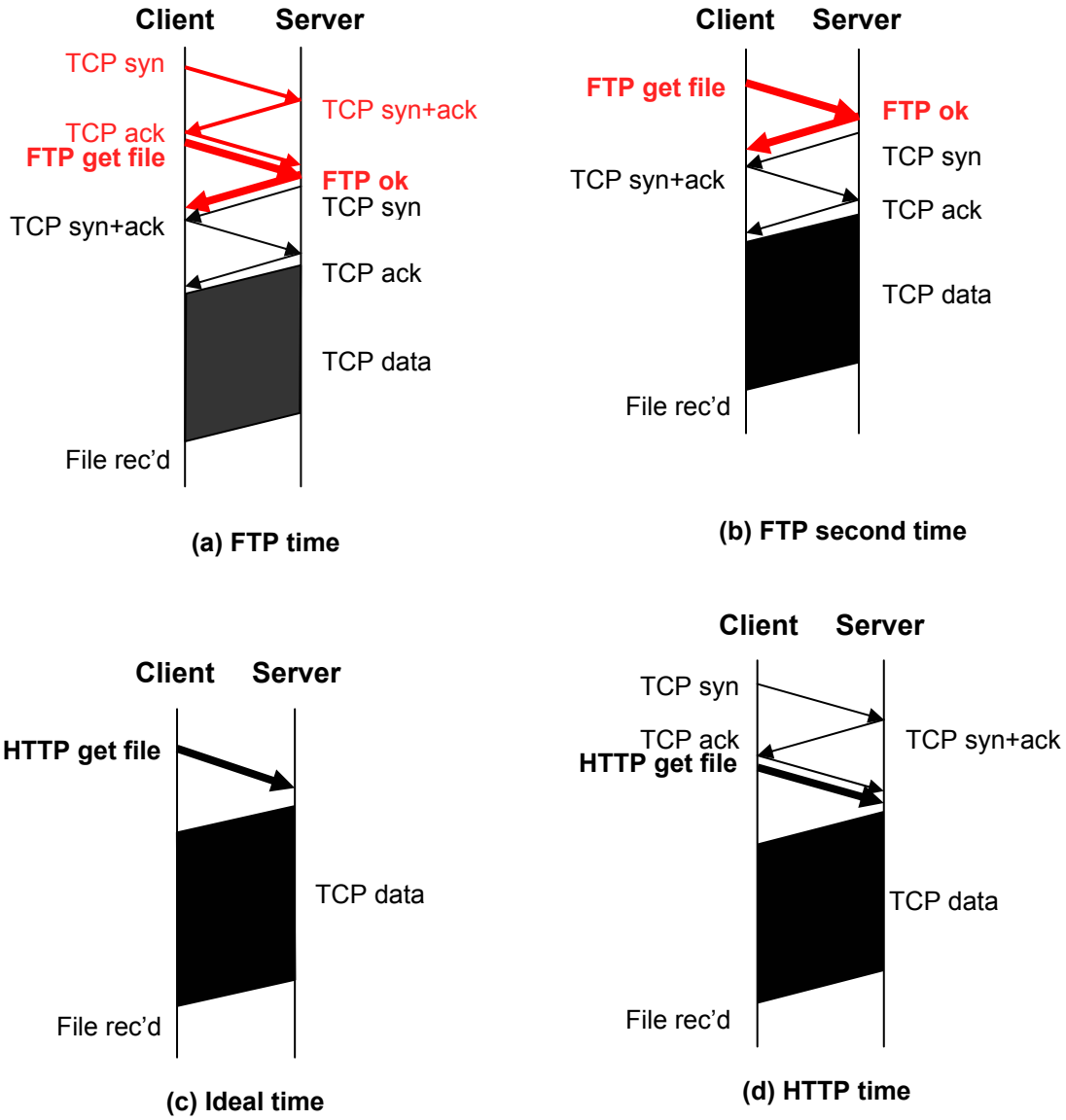

Figure 1.2: Comparison of Round Trip Times of FTP and HTTP

In Figure 1.2(a) we see that for the first FTP transaction, different TCP connections are set up between the client and the server for the request and the response. Note that the acknowledgements travel on the same connection. Hence the overall time

passed between the request leaving the client and the client beginning to receive the response is the equivalent of 3 RTTs of overhead. In Figure 1.2(b) which is a subsequent FTP request between the same client and the same server, the request is sent on an existing connection whereas for the response a different TCP connection is set up. Hence the effective overhead is just 2 RTTs. Figure 1.2(c) shows us the ideal scenario which we want to get as close to as possible. In this case there is just one RTT of overhead which is inevitable. Figure 1.2(d) shows us a first HTTP/1.0 transaction which requires only two round trips, after the request has been issued from the client to the server, before the arrival of the response at the client. This is because, the HTTP/1.0 client does not tear down the initial connection, which is set up when it sends the request to the server. The HTTP/1.0 server does not need to set up a new connection to send the response and sends the response on the same connection as the request. Because of this, one round trip involving the TCP triple handshake for connection establishment is avoided. Hence HTTP/1.0 effectively requires one RTT less than FTP for all transactions.

2.1.2 Issues Using HTTP/1.0

Even though transactions between HTTP/1.0 clients and servers require 1 RTT less than FTP, objects are downloaded from server to the client with each object requiring a separate HTTP over TCP connection. This method is not suitable for a network characterized by the presence of a high delay link, such as a satellite link. This is because, most of the time, multiple objects are downloaded back-to-back. Depending on the latency of the network and/or link, this results in a significant delay before the next request arrives at the server. This implies that when the request for a certain web-page is sent to the web-server, the base HTML page is sent to the client first and the existing

connection is torn down. Once it is parsed, the client sets up a new connection with the server to request the first embedded object in the page. Once the client receives the first embedded object completely and the connection is torn down, the client begins to setup another connection with the server in order to retrieve the next embedded object. This continues till all the objects are received by the client [4]. These days most web-pages comprise of several embedded objects. Thus the additional overhead of setting up a new connection for fetching each object individually has an adverse effect on how much time it takes to fetch the entire web page. This delay is all the more significant in our case, as the satellite link latency makes the perceived latency at the web-browser's end very large.

2.2 HTTP/1.1

HTTP/1.1 was developed to eliminate the setup overhead which came with HTTP/1.0. This is done by incorporating pipelining of requests [5]. "Pipelining" means clients that support HTTP/1.1 allow multiple requests to be written out to a socket one behind the other without waiting for the corresponding responses. The client then waits for the responses to arrive in the order in which they were requested.

2.2.1 Advantages using HTTP/1.1

The actual latency involved with the transfer of each and every embedded object in the web-page is drastically reduced by using HTTP/1.1 compatible clients and servers. The "pipelining" of requests results in a dramatic improvement in page loading times, especially over high delay connections [6]. Pipelining does not require objects on a web page to be downloaded back-to-back with each object requiring a separate HTTP

connection. All the objects embedded on the webpage are pipelined and can be sent over even a single HTTP connection.

Pipelining also dramatically reduces the number of TCP/IP packets by eliminating the setup overhead and hence leading to more efficient utilization of the available bandwidth. Note that these differences mentioned above and others are very well summarized in [7].

2.2.2 Issues using HTTP/1.1

HTTP/1.1 is very widely deployed and most clients and servers are compliant with it. Unfortunately, we note that, being HTTP/1.1 compliant does not mean implementing the pipelining feature. HTTP/1.1 compliance implies only support for persistent connections. None of the versions of the most popular browsers, namely, Netscape and Internet Explorer, have actually implemented pipelining [8]. The reason for this is non-intuitive as it would seem that there is a lot to benefit from this scheme and not implementing it would not be wise. The reason for is that HTTP/1.1 clients, more often than not, do not get assurance from the HTTP/1.1 servers that the pipelined requests/responses will not be aborted by a premature connection close. This assurance can be given to the client/server by specifying the content length of the pipeline in the “content-length” field in the HTTP header. The end-of-data can be specified in 3 ways [9]:

1. Content-Length field – This requires a lot of data copying on the server
2. Content-Type field – This specifies a "boundary-delimiter". This is inefficient as the server has to inspect every byte of data
3. The server (not the client) closing the connection

If the client does not have the assurance that the connection will not close prematurely, it will decide to play it safe and not pipeline objects on the connection. This way, if the connection does terminate after the first object goes through, other packets, which would have otherwise followed, will not be lost. This means that the SS algorithm does not kick in and the throughput of the system does not drop. This is the reason that browsers are hesitant to implement HTTP/1.1 pipelining.

2.3 p-HTTP

p-HTTP stands for "persistent" HTTP. This is not a HTTP version but is actually a development of the protocol. Persistent HTTP means that the HTTP connections established on TCP connections are not torn down immediately after the transfer of data has occurred. The connections that were setup for a previous request-response interaction are maintained for a certain amount of time. The motivation for this development comes from the fact that HTTP requests tend to repeat over time.

2.3.1 Advantages using p-HTTP

Since persistent connections anticipate repeated requests, the same connection is re-used when another request for the same resource or another resource from the same source is ready. This eliminates the setup time required to establish the TCP connection all over again. This too leads to a reduction in the number of packets and we hence have higher throughput [4].

Another advantage of using persistent HTTP connections is the fact that the HTTP client or server will only pipeline after having gotten one request through a connection which is persistent. This is because the HTTP client/server does not know

whether the other end supports persistent connections. After that first request, which handles the negotiation it is known for sure whether the other end supports persistent connections and hence pipelining is initiated. Thus persistent connections assist pipelining which leads to drastically reduced latency as discussed in section 2.2.1.

Note that HTTP/1.1 in conjunction with TCP uses "TIME_WAIT" states of connections to establish long-lived connections. A TIME_WAIT state is information about a connection which is maintained by a client or a server even after the connection is torn down, just in case a similar connection is required again. This information is maintained for up to 4 minutes. Using these TIME_WAIT states obviates the need of the TCP triple handshake for connection establishment if a similar connection is required within 4 minutes of the earlier connection being torn down [4]. The drawback of this scheme is that too many of these states could fill up the space in the registers allocated for saving the state of the connections. This makes spawning new connections impossible till some of these TIME_WAIT states expire. Another advantage of p-HTTP connections is that, they do not require such TIME_WAIT states and hence do not lead to such problems.

2.3.2 Issues using p-HTTP

The problem with p-HTTP is that, by itself, this development does not substantially affect web access speeds on slower networks. This is because we need access speeds greater than 200 Kbps for at least a 50% reduction in latency. This kind of a reduction in latency is essential for the end user to perceive the improvement in performance. Typically, the overall improvement is only 11% reduction in latency as

non-broadband web-access speeds are slower than 200 Kbps and are approximately around 17 Kbps [2].

Another disadvantage of p-HTTP is that the complexity at the application layer increases. This is because segmentation and reassembly of packets is now the responsibility of that layer i.e. the application layer [4].

2.4 Conditional Statements

Conditional statements are HTTP requests which do not necessarily ask for the entire response from the server. A conditional method requests that the response be transferred only under the circumstances described by the conditional header field(s). HTTP allows the client to ask the server in a single query whether the resource has changed since it was last accessed by it (the client). If the resource has changed, the client gets the new version. If it hasn't, the server just tells the client that it hasn't changed and sends nothing. This mechanism is called “Conditional GET”. These conditional statements typically request partial responses and are used with some form of cache validation. The aim of conditional statements is to reduce unnecessary network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring data already held by the client [5].

2.4.1 Advantages using Conditional Statements

Conditional statements reduce the latency in a HTTP transfer simply by reducing the number of bytes transferred. For example, using a conditional GET statement instead of a regular GET statement along with appropriate cache can save up to 90% of the data from being transferred [9]. A regular GET command gets just one object from the page.

If we use a GETALL statement instead, we receive all the embedded objects in the page, continuously over one connection. The ALL option is added using a pragma. A pragma is a general header-field that includes implementation-specific directives for the client, server and any intermediate proxies or caches [5]. The GETALL statement is useful only when no images from that page are cached and it is only used for the first time the page is requested from a web-server by a client.

The GETLIST command is used if the client does not know what it is actually getting. The response to this command lists all the objects on the web-page. This lets the client select what it does not have cached and hence eliminates the retransmission of existing objects [9][10].

The "partial GET" request message includes a range header field. A partial GET requests only a specific part of the entity to be transferred. This method is intended to reduce unnecessary network usage by allowing partially-retrieved entities to be completed without transferring data already held by the client.

2.4.2 Issues using Conditional Statements

Even though the reduction in the latency by reducing the number of bytes transferred is very useful in a link with high latency, such as a satellite link, a multitude of options in the conditional requests lead to the lack of an optimal algorithm. Also, an efficient cache replacement algorithm does not exist for such a scheme. This may hence lead to cache-bursting.

3 Other Schemes and Developments

In this chapter we investigate the developments undertaken to solve the problem of user-perceived latency in high delay networks. Here, we have studied schemes that are not only modifications of HTTP but also some that are independent of the HTTP protocol. Such developments are changes made to other aspects of the network such as the transport layer protocol, the content on the WWW, compression and encoding of data and transportation methods of the data.

3.1 Changing the Transport Layer Protocol

After the observations we made in the previous chapter, the most obvious question we try to answer is whether there is another transport layer protocol that HTTP performs better over for high delay links. In this section we look at modifications to the underlying transport layer protocol. In the sub-sections that follow, we look into the usage of schemes that save the state of the connections.

3.1.1 Advantages and Disadvantages of T/TCP and S-TCB

Since TCP is the most popular transport layer protocol used we look at different modifications made to TCP in order to reduce the user-perceived latency. TCP is a connection-oriented protocol and is not ideal for the request-response nature of the HTTP protocol which gives rise to bursty Internet traffic. Note that the main sources of HTTP increased actual delay, along with the perceived delay using TCP as the transport layer protocol are:

- the size of the protocol headers
- the triple handshake for connection setup
- the slow start algorithm for congestion avoidance and retransmissions and
- the congestion control delay due to packet loss

We do not look at changes to TCP, which make TCP as a protocol, satellite friendly. This, as mentioned earlier is out of the scope of this thesis. Hence developments such as TCP - Selective Acknowledgements (SACK), Delayed Acknowledgements and different congestion avoidance schemes are outside the scope of this thesis.

Transaction TCP (T/TCP) when used instead of TCP as the transport layer protocol, caches the state information of the connection, such as the RTT and control block parameters. Due to this, the next time a connection needs to be setup between the client and the server, the SS algorithm is quicker than usual. This is because the client and the server recognize each other hence permitting early delivery of packets [4][11]. This reduces the user-perceived delay.

S-TCB (Shared TCP Control Block) functions on the same principle as T/TCP except that S-TCB can be used for both concurrent as well as serial connections, whereas T/TCP can be used only for serial connections.

HTTP behavior was tested over TCP, which establishes a new connection for each request and over T/TCP, which caches state information and hence avoids the three-way TCP handshake delay during connection establishment. HTTP behavior was also tested over UDP based protocols, which avoid TCP setup costs and p-TCP (persistent TCP), which is very similar to p-HTTP and has no connection setup delay and a testing model was proposed. Even though this model was not ideal for high delay-bandwidth

product links, it was found that connection caching protocols are very useful in terms of reducing the apparent delay at the user's end. This observation was true even for single web-page hits which were not repeated [12].

The most significant drawback of changing the transport layer protocol scheme is that it is a "change the world" approach. This approach is hence not very practical.

3.1.2 HTTP State Management Protocol

In this section we take a look at the variations in saving the state of the connections at the application layer and the possible impact on the delay experienced by the user while browsing the Internet. We also look at the impact on the security and privacy of the user in this scheme.

The HTTP State Management facility [11] does provide an increase in functionality over ordinary HTTP and HTML. It lets us save state between HTTP transactions. This additional functionality includes the ability to exchange URLs between users, of resources accessed during stateful sessions, without leaking the state information associated with those sessions. This leads to a significant reduction in the delay in setting up these connections. The ability to maintain session state without "cache-busting" is also made possible with this functionality. This means that, separating the session state from the URL allows a web cache to maintain only a single copy of the named resource. If the state is maintained in session-specific URLs, the cache would likely have to maintain several identical copies of the resource.

Note that this state management facility is also able to implement sessions with minimal server configuration and minimal protocol overhead, as compared to other techniques of maintaining session state. This is again a very important feature when we

look at our scenario of high delay satellite channels. This state management facility also has the ability to associate the user with session state whenever a user accesses the service, regardless of where the user enters. This means that the user will be associated with the session state irrespective of whether the user enters through a particular home page or portal. The saved sessions or connection information can be maintained across client invocations, system reboots, and client or system crashes [11].

The most significant drawbacks of this variation of saving the state of the connections scheme are issues concerning the privacy and security of the user. Hence many of the uses of this scheme are not recommended by the IETF (Internet Engineering Task Force) or are believed to be harmful and are discouraged. For example, state information can be transmitted from the service to the user by embedding a session identifier in one or more URLs which appear in HTTP redirects, or dynamically generated HTML; and the state information may be returned from the user to the service when such URLs appear in a GET or POST request. HTML forms can also be used to pass state information from the service to the user and back, without the user being aware of this happening [11]. This compromises the privacy of the user.

It hence becomes very essential to realize that the use of HTTP State Management is appropriate whenever it is desirable to maintain state between a user and a service across multiple HTTP transactions, provided that:

- The user is aware that session state is being maintained and consents to it.
- The user has the ability to delete the state associated with such a session at any time.

- The session information itself cannot contain sensitive information and cannot be used to obtain sensitive information that is not otherwise available to an eavesdropper.

Also, the information obtained through the ability to track the user's usage of the service or the user's browsing habits is not disclosed to other parties without the user's explicit consent. It is also inappropriate to use the HTTP state management protocol as an authentication mechanism.

3.2 Changing the Nature of Web-Content

One of the ways to reduce the user's perceived latency is by actually reducing the number of bytes transferred over the space segment. Web pages are getting more and more complicated with not only the number of embedded objects per page increasing, but also the relative size of each embedded object is increasing [5] [13]. We examine the possibility of changing the nature of content on the WWW to make it more suitable for transfers on a high delay link.

Nielsen et al suggest that changing web-content suitably will help reduce the size of the files on the WWW and hence make them more suitable for transfers over satellite links [14]. Most web-pages use Graphic Interchange Format (GIF) or JPEG (Joint Photographic Experts Group) format (JPG) for graphics or images on web-pages. These formats, although pre-compressed are large in size. The use of PNG (Portable Network Graphics) instead of JPG or GIF images helps improve performance as this format is designed to be of a much smaller size. Also using CSS (Cascading Style Sheets) will help improve network performance in terms of latency because these style sheets too are designed to be much smaller in size.

Also note that the content encoding of only the body of the HTTP message but not the headers will be useful as HTTP dominates a considerable portion of Internet traffic.

This scheme shares the same drawback with the scheme mentioned in section 3.1, i.e. it is difficult to implement and requires a universal change before the benefits of the scheme can be seen.

3.3 Data Compression

Most browsers and web-servers incorporate some content compression and most browsers can perform streaming decompression of content that is compressed using a utility such as Gzip [15]. Data Compression is an effort towards reducing the number of bytes transferred over the satellite segment since reducing the number of bytes transferred over the network will lead to a reduction in the transfer time which will lead to a reduction in the perceived latency at the user's end. Resources that are obtained from a web-server in response to a client's HTTP request typically contain HTML, JavaScript and XML documents which are basically ASCII test files. These files contain many repeated sequences of identical information and hence they can benefit immensely from compression. Other objects on the web-page such as JPEG images, GIF images, audio and video are pre-compressed and hence would not benefit from further compression. Lossless compression of text files has shown up to a 27% reduction in transferred resources from the web-server to the client [16].

Simple lossless algorithms can be used to achieve outstanding results in terms of the compression ratio. The "compression ratio" is defined as the ratio of the size of the original resource to the size of the compressed resource [17]. Lossless compression must

be used since the alteration of even one bit in the base web-page could change the entire web-page. The most popular algorithms are Huffman encoding and the Lempel-Ziv algorithm.

Huffman encoding works on the principle of assigning a code to each of the repeated symbols in a file. The most frequently occurring symbols are given the shortest code and the least frequenty occurring symbols are given the longest code. The Lempel-Ziv algorithm works on the principle of creating a directory of sequences. Each time a sequence that has already occurred before repeats, it is replaced with a pointer to that previously occurred sequence along with the length of repetition [18].

We note that modems currently implement compression techniques. Unfortunately these are neither optimal nor as effective as HTTP compression. This is because modem compression works with only small blocks of fixed length data and not with a large portion of a file as is the case with HTTP compression. Also, modems offer only point-to-point compression [16].

HTTP/1.0 incorporates the "Content-Encoding" entity which lets the client understand what transformation or compression has been used on the arriving resource. Content-Encoding applies only to end-to-end compression [3].

HTTP/1.1 has extended support for compression by adding "Transfer-Coding". The Accept Encoding is better defined for HTTP/1.1. Here the transfer-coding values are a property of the message and not of the entity. Due to this, transfer-encoding values are used to indicate the hop-by-hop compression or encoding that has been applied to the resource [5].

Using Zlib for HTML compression is much better than Modem Compression. This mitigates SS for small responses by reduction of the number of packets to be transferred and reduces the number of bytes for large responses. Data compression along with T/TCP amounts to enormous savings of time and bandwidth [4] [19]. We believe that it may help to decrease the user-perceived latency if the HTTP headers in the response are compressed using a simple compression utility such as Gzip.

The biggest drawback of data compression in this application is the fact that proxy support for the "vary" header does not exist. This is necessary for a proxy cache to store and handle compressed resources correctly. Hence, even though a given resource is compressed by a large factor, the effectiveness of this compression is negated if the server has to constantly retransmit this compressed resource version over the space segment because the proxy cache could not serve this request. Another drawback is the lack of a standardized algorithm for data compression.

3.4 Using a larger TCP Congestion Window (CWND)

Intuitively, it is obvious that using a larger TCP congestion window size (cwnd) will help in reducing the latency, especially so in an environment where the link is characterized by high delay. An experiment was performed using a larger initial cwnd in TCP. Note that a regular cwnd is 1 segment i.e. 1 KB and a larger initial cwnd implies 4 segments, i.e. roughly 4 KB [20]. Thus:

$$W_i = min(4.MSS, max(2*MSS, 4380)),$$

where, W_i is the initial window size and MSS is the Maximum Segment Size.

Equivalently, the upper bound for the initial window size is based on the MSS, as follows [20]:

if (MSS $\leq$ 1095 bytes)
    then $W_i \leq 4 * MSS$;
if (1095 bytes < MSS < 2190 bytes)
    then $W_i \leq 4380$;
if (2190 bytes $\leq$ MSS)
    then $W_i \leq 2 * MSS$;

3.4.1 Advantages of a Larger Initial TCP Window Size

There are several advantages in having an inflated initial congestion window for TCP. By inflated, we mean four segments instead of just one segment. The advantages of a 4KB window size are listed below.

- When the initial window is one segment, i.e. 1 KB, a client employing delayed acknowledgements (ACKs) is forced to wait for a timeout before generating an ACK. With an initial window of at least two segments, the receiver will generate an ACK after the second data segment arrives. This eliminates the wait on the timeout (often up to 200 ms on the satellite space segment).
- For connections transmitting only a small amount of data, a larger initial window reduces the transmission time (assuming at most, moderate segment drop rates). For HTTP transfers that are less than 4KB, the larger initial window would reduce the data transfer time to a single round trip time (RTT).
- For connections that will be able to use large congestion windows, this modification eliminates up to three RTTs and a delayed ACK timeout during the

initial slow-start phase. This would be of particular benefit for high-bandwidth large-propagation-delay TCP connections, such as those over satellite links.

3.4.2 Disadvantages of Larger TCP Initial Window Size

Increasing the initial congestion window size is not free of drawbacks. TCP assumes that packets that have been dropped have been dropped simply because of impending congestion. Note that the satellite space segment has a higher Bit Error Rate (BER) than most other channels. In this case if packets are dropped because of corruption, TCP, by default, assumes that it is because of congestion.

In terms of the potential for congestion collapse, we consider two separate potential problems for the network. The first problem would be a scenario where a large number of packets on congested links were duplicate packets that had already been received at the receiver. The second problem would be a scenario where a large number of segments on congested links were segments that would be dropped later in the network before reaching their final destination. In terms of the negative effect on other traffic in the network, a potential disadvantage of larger initial windows would be that they increase the general packet drop rate in the network [21].

Note that bit errors on the satellite segment imply an immediate reduction in cwnd size. Since in smaller sized transfers, the cwnd is already small, reducing it any further does not make an appreciable difference i.e. bit errors do not cause a drastic increase in latency. They do so only in large resource transfers, where the cwnd is initially large. Hence for large web-page transfers, we must take care to reduce the bit error rate, as any packet loss is attributed to congestion, even if it is due to corruption, by

TCP. Hence it is crucial that one should proceed with great caution while increasing the congestion window size on a channel with a high BER.

Also, a larger cwnd helps in reducing time by one to 1.5 RTTs and is hence better for those links where the RTT is high, i.e. networks with a high delay-bandwidth product segment. This can have adverse effects on the terrestrial TCP link performance [22].

3.5 Cache Validation and Delta Encoding

Web resources change over time due to several reasons. They may change due to a change in the content of the web-page. For example, a news web-site must change or update itself as frequently as possible to keep the visitor informed with the latest happenings. Sometimes web-pages change even if the content of the web-page has not changed. These cosmetic changes are made to the web-pages in order to give them a fresh or current look as noted in [12]. Assuming that the client has visited a certain web-page earlier and has it cached, it would make sense to cross check using the timestamp of the cached resource whether it is still fresh, i.e. if the web-page has changed since the time the client visited it or not. The client hence sends the timestamp of the cached resource with its subsequent request for the same resource to the web-server. If the web-page has not changed since the time the client accessed it last i.e. timestamp, the server responds with a "Not-modified-since" instead of re-sending the resource to the client. This response tells the client that its cached version is still fresh and the browser displays that page to the user. This process results in huge savings in bandwidth utilization. This is especially important for satellite links as not re-sending the web-resource and sending of a relatively negligible sized response results in a steep reduction in user-perceived latency at the client's end. This process is Cache Validation.

What if the resource has changed in the meanwhile? It would be better to send a record of changes in the web-page over the satellite link instead of sending the entire changed resource if the changes are minor changes like the cosmetic changes mentioned above. This is because the record of changes would typically be much smaller in size than the entire changed web-page for a change like this. Once the client knows what changes have been made to the web-page, it can generate the new web-page locally and paint it on the browser. It is important to note here that currently, network speeds are relatively much slower than processor speeds.

A "delta" is a file that contains the information about the changes in a web-resource as compared to another instance of the same web resource that is already present in the client's cache. We transmit only the deltas i.e. the difference in the webpage when it is being accessed and when it was accessed last. The deltas, on an average, have been seen to be much smaller than the page or even page objects and hence the transfer time over a high delay-bandwidth product link is minimal [19] [23]. The current page can hence be reconstructed using the previous cached page and the incoming delta – using Delta Encoding.

3.5.1 Benefits of Delta Encoding

Both the Delta Encoding and Data Compression methods aim to reduce the latency by reducing the number of bytes transferred on the space segment. They differ drastically in the sense that a compressed version of the resource is good enough for the client to re-generate the resource and display it for the user. The same does not hold true for deltas. The resource cannot be generated using a delta alone because it needs the

cached version of the resource relative to which the delta has been generated. Hence delta encoding is always used in conjunction with cache validation.

When delta encoding is used along with data compression, up to 83% savings in latency can be achieved [23]. Different delta generators are as follows:

- diff-e: A compact format generated by the UNIX diff command
- compressed diff-e: The above plus compression using Gzip
- vdelta: A good delta algorithm which works best among the ones mentioned in this list.

A big advantage is that delta generators are easy to implement as they exist as library functions.

Another benefit of delta encoding is that the size of the deltas can be further reduced by compressing them. A compression scheme such as Gzip may be used.

Deltas could also be cached at the servers instead of generating them on the fly as requests with timestamps arrive. This ensures quicker responses. Proxies too could be used for caching and generating deltas locally instead of at the server.

Another useful application of delta encoding could be the clustering of URLs. This scheme involves calculating the delta between URL requests. It turns out that this scheme is very useful since many subsequent requests are made for resources with the same URL prefix [23].

Delta encoding is hence strongly encouraged as it leads to huge bandwidth savings, resulting in reduction of latency [24].

3.5.2 Drawbacks of Delta Encoding

In spite of all of its benefits and advantages, delta encoding suffers from some non-trivial limitations. Primarily, delta encoding is useful only if a certain web-page is accessed multiple times. Thus, the usage of deltas depends upon whether the resource is "delta-eligible" or not. To be delta-eligible the resource has to be a repeat requested resource. Note that on an average 25% of Internet traffic is delta-eligible.

For the benefit of delta encoding to be appreciated, a delta must me at most less than half the original size of the object.

Another important drawback is that to calculate deltas, the web-server has to cache various versions of updates of a resource. The question that arises is, how many versions or "instances" does one keep cached at any given point? Also, for how long does the cache keep these instances? These questions have no straightforward answer as of now.

Delta encoding has implementation issues with HTTP/1.1 [24]. Even though HTTP/1.0 uses the "if-modified-since" command to get deltas, HTTP/1.1 uses "all-or-none" in the above aspect. The "if-modified-since" is used for a "conditional GET" in HTTP/1.1. The following features are hence required in HTTP/1.1:

- A method to mark the request as conditional
- A method to specify which delta will be applied to the client's request
- A method to show that client can support a specific delta (vdelta/diff -e etc.)
- A method to mark that a response is delta encoded in a certain format

All of the above are required for HTTP/1.1 compatibility with delta encoding.

Compression of the deltas also suffers from the drawback that the compression algorithm is not yet standardized. Also, the repeated compression of deltas must be avoided as compression consumes time and processor cycles.

Last but not the least, recall that timestamps are used for cache validation and for selecting which delta to respond with when pre-creation of deltas is used. Serious errors can occur due to lack of timestamp resolution or loss of synchronization between the client and the server.

4 Proposed Scheme I – Optimized Browser

We have provided a literature review, which included related work and proposed schemes, in the previous two chapters. We now describe a scheme called the "Optimized Browser" in this chapter to reduce the latency for Internet content delivery over satellite links and making the process of Internet browsing, satellite friendly. This chapter is roughly divided into two parts. The first part, i.e. section 4.1 through section 4.5, describe the initial "Adaptive Connections" scheme and the latter part of the chapter i.e. sections 4.6 through 4.8 describe the evolution of this scheme to the optimized browser scheme.

4.1 Introduction and Concept

The motivation for this scheme comes from the fact that both the popular and most recent versions of HTTP request web resources primarily back-to-back. The difference being that HTTP/1.0 requires separate connections for each object requested back-to-back, while HTTP/1.1 requests all objects back-to-back on just one connection hence eliminating the multiple RTTs worth of connection setup overhead.

The next step would be to try and achieve a simultaneous gathering of multiple objects from a web-page, thus allowing for "burst" i.e. sharing of available bandwidth depending on the number of active and inactive users. Intuitively, the scheme will have better performance than that of HTTP/1.1 and obviously much better than that of HTTP/1.0. This is due to the fact that the triple-handshake for connection establishment can either be completely avoided for the connections or the handshake delay for all of the parallel connections between the client and the user overlaps. Hence the effective

handshake delay time is just the equivalent of one triple handshake for connection establishment. Also note that one can intuitively estimate the amount of time required for the web-page to load fully, assuming that the number of connections setup is actually equal to one more than the number of embedded objects in the base page. This time would be equal to the time required for the largest element in the web-page to be transferred from the server to the client, assuming that burst of that magnitude is possible. Also, note that multiple parallel connections implies that each of the connections that are setup are independent of each other. The advantage of this independence is that the initiation of the transfer of one object does not depend on the completion of the transfer of another embedded object.

The initial scheme we thought of was one where the number of connections and the initial congestion window size of each connection would depend on the number and nature of objects embedded in the web-page. Hence the scheme was named "Adaptive Connections".

"Adaptive" means the nature and number of connections between the client and server/proxy vary depending on the properties of the web-page requested. Say, the web-page is plain HTML, or has a single large image, a single TCP connection with a large CWND would be ideal. If the web-page has several moderate-sized elements, it would be best to use multiple connections with a small CWND size, as the set-up time for each connection would be the same (and would happen concurrently).

After our experiments, we realized that, the performance of the scheme would benefit more if the number of connections were actually optimized, depending on the utilization of the space segment and the state of congestion in the network. We hence

renamed this evolved scheme and called it the optimized browser scheme. The details of this transition are mentioned in the implementation section.

4.2 Implementation

4.2.1 Simulation Setup

We decided to use the OPNET 8 modeler. We used a simple model of one client and one server as shown in Figure 4.1 below.

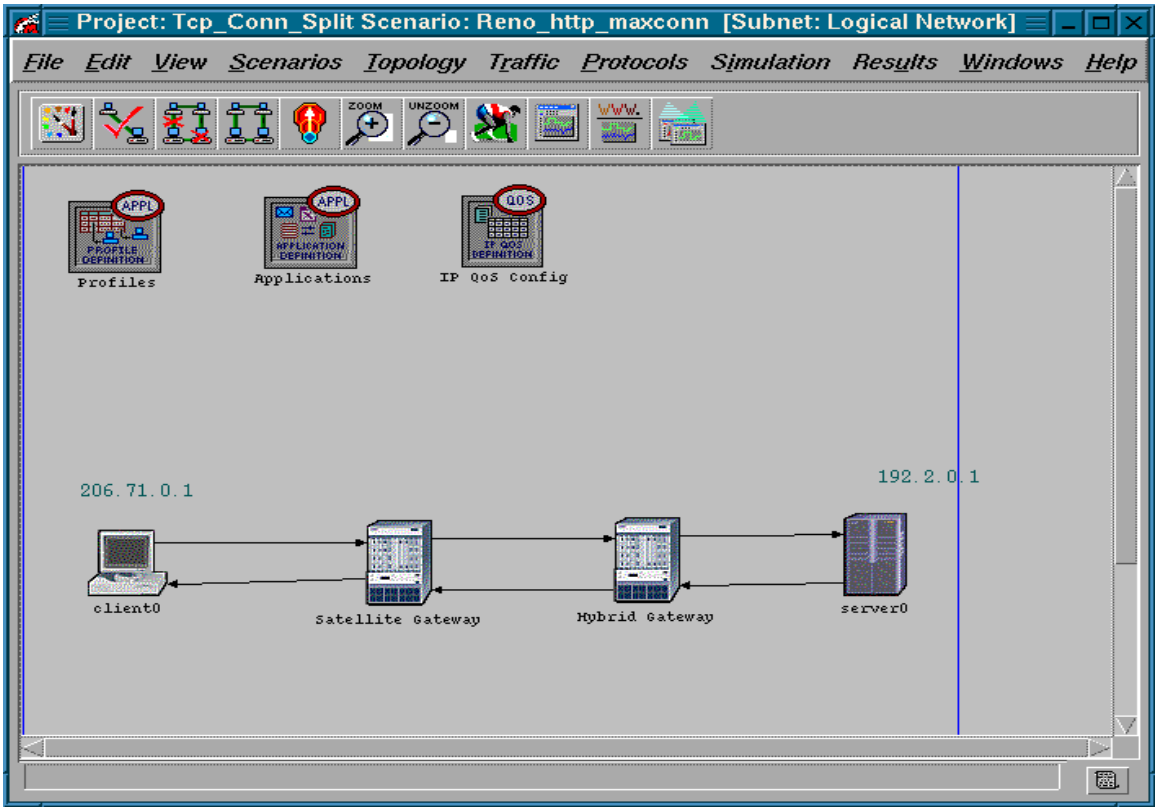

Figure 4.1: Setup of Experiments

As can be seen in the window of the OPNET Modeler 8 above, “client0” is the client, “server0” is the web-server. The “satellite gateway” is aboard the satellite and the “hybrid gateway” is at the NOC. The space segment was simulated in this setup by incorporating

a BER of 10^{-8} and a latency of 0.125 ms in each of the four segments between the client, the satellite and the hybrid gateway. This is the space segment. The network between the NOC and the web-server is the ground segment.

We simulated FTP, HTTP/1.0, HTTP/1.1, connections with a large initial cwnd and multiple parallel connections on this setup. The FTP, HTTP/1.0, HTTP/1.1 were simulated by changing the parameters of the application layer protocol within the OPNET modules.

Since TCP Reno is the most popular version of TCP used, we implemented it on the ground segment in our setup. On the space segment we used a protocol called Receiver Window Backpressure Protocol (RWBP) [25] [26]. This is a protocol that is derived from TCP Reno but it is more suited for WWW transactions over high-delay links. This is because it does not have TCP's Slow Start and Congestion Avoidance algorithms for congestion control. The connection between the client and the server was hence a "split-connection" TCP-RWBP connection.

4.2.2 Test Scenarios

For the testing and comparison of the performance of FTP, HTTP/1.0, HTTP/1.1, a connection with a larger initial cwnd and multiple parallel connections we generated several web-pages with different characteristics. The characteristics that were varied were the number of embedded objects in the web-page and their sizes. OPNET has pre-defined definitions of small, medium and large size objects on a web-page. The size range of a small object is 500 bytes. Similarly, the size range of a medium-sized object is 5 Kbytes and that of a large-sized object is 50 Kbytes. Several web-pages were created including

extreme cases such as a web page with just one extremely large object and another web page with about fifty small objects. 12 other web-pages were created by varying the number of objects and their sizes respectively. A real-life web page i.e. “www.espn.com” was modeled by replicating the exact number of objects on that page and their exact sizes in OPNET.

We then set up different “scenarios” for the testing of each of the above application layer protocols in terms of their performance with respect to latency, in transferring the web pages mentioned above. Each scenario refers to a specific application layer protocol/method. The different scenarios simulated were FTP, HTTP/1.0, HTTP/1.1, a single connection with a large initial congestion window and multiple parallel connections with a maximum number threshold of 51. Each of these multiple parallel connections have a regular cwnd size of 1 segment. This was the basis of defining the scenario mentioned below.

The “MaxConn” scenario is defined as a scenario where the maximum number of HTTP over TCP connections that can be set up between the client and the server, is equal to the number of embedded objects in the web page. In our setup, we arbitrarily set this to be 50, as the largest page we simulated has 50 elements. The TCP SS starts with 1 segment (regular cwnd).

The “ExactConn” scenario, on the other hand, is defined as exactly one connection with a large cwnd i.e. the SS algorithm starts with 4 segments instead of just 1.

The idea of MaxConn was to reduce the latency in the network, by getting exactly one element per connection. Since, no pipelining or persistence was required; we used

HTTP/1.0 for these connections. Theoretically, HTTP/1.0 can support a large number of connections.

We initially intended to use the meta-information field in the HTTP response header to indicate the number of embedded objects in a page. This information could also be pre-fetched. Based on this information, the client decides how many connections it wants to set up with the server for the transfer of data, depending on the nature of the web page. If the web page has several small to moderate sized objects, we tested the use of multiple connections with a regular cwnd size. We expected that the connection set-up time for each connection would not contribute multiple times to the latency since the setup would happen concurrently [20]. Hence the maximum time for the entire page data to transfer to the client, in this case, would then be equal to the time required for the single largest element on the page to transfer and this scheme might prove to be beneficial. A larger cwnd helps in reducing time by 1-1.5 RTT and is hence better for those links where RTT is high, i.e. networks with a high delay-bandwidth product segment, for pages with large objects.

In this implementation we assumed that the server knows the number of embedded images in the page beforehand. We did this to validate our conjecture that there would be an improvement in performance using this method, and then to follow up with the idea of exploiting the meta-data that we can get about the requested web page through the HTTP response headers.

4.3 Experiments

Experiments were first run using the generated web pages and FTP, HTTP/1.0 and HTTP/1.1 as we wanted to validate the accuracy of our simulations. We hence used the first three application layer protocols not only for a sanity check but also as a benchmark to test the performance of the other schemes. We collected statistics for several parameters in these simulations. We observed the active connection count, traffic sent and received over the link in *packets/sec* and *bytes/sec*, object and page response times as the global statistics. We also observed the link utilization, the throughput (*bits/sec*) and the queuing delay as the object statistics.

The first set of experiments was run using FTP as the application layer protocol. The second set of experiments was run using HTTP/1.0 as the application layer request-response protocol. For these cases, the simulations, where the client requested the page from the server and this transfer was completed, were then run and the desired statistics were collected. The existing scenario was then duplicated for the third set of experiments and the application layer protocol changed to HTTP/1.1. The simulations were run again. In all the above three experiments, we tested all of the generated web pages with each of the application layer protocols.

In the fourth and fifth set of experiments we used our proposed application layer schemes, i.e. “ExactConn” and “MaxConn” instead of FTP, HTTP/1.0 or HTTP/1.1 respectively. All the generated web pages were requested and then transferred from the web server to the client. Modifying the “profiles” module, in the simulation setup, accomplished this.

4.4 Observations and Results

The statistics from the three different scenarios of the simulation runs of the first, second and third set of experiments were compared. This was done to create a benchmark for the comparison of our schemes and for validating our test-bed. The results stated in the literature survey were validated.

Figure 4.1 shows the actual number of connections that were spawned and their durations. This refers to the experiment where the ESPN web-page which had 50 embedded objects, was transferred from the server to the client using all the generated scenarios.

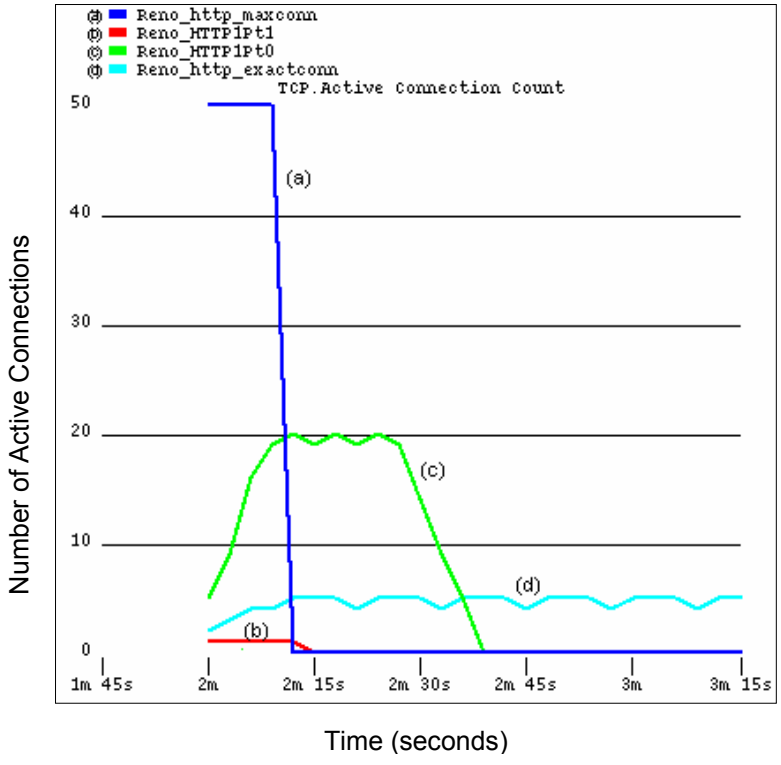

Figure 4.1: Active Connection Count of all the Scenarios

Several observations were made with respect to the results of the above experiments using different scenarios depending on which (generated) web page was used for the transfer.

In the experiment, with the ESPN page, we observed (refer Figure 4.2) that the traffic sent and traffic received per second (*bytes/sec* and *packets/sec*) was highest for MaxConn and considerably lower for HTTP/1.1. Both scenarios required very little time for the entire transfer, though. The time required for this transfer using HTTP/1.0 was much higher. The time required for the transfer was also extremely high when using the ExactConn scenario.

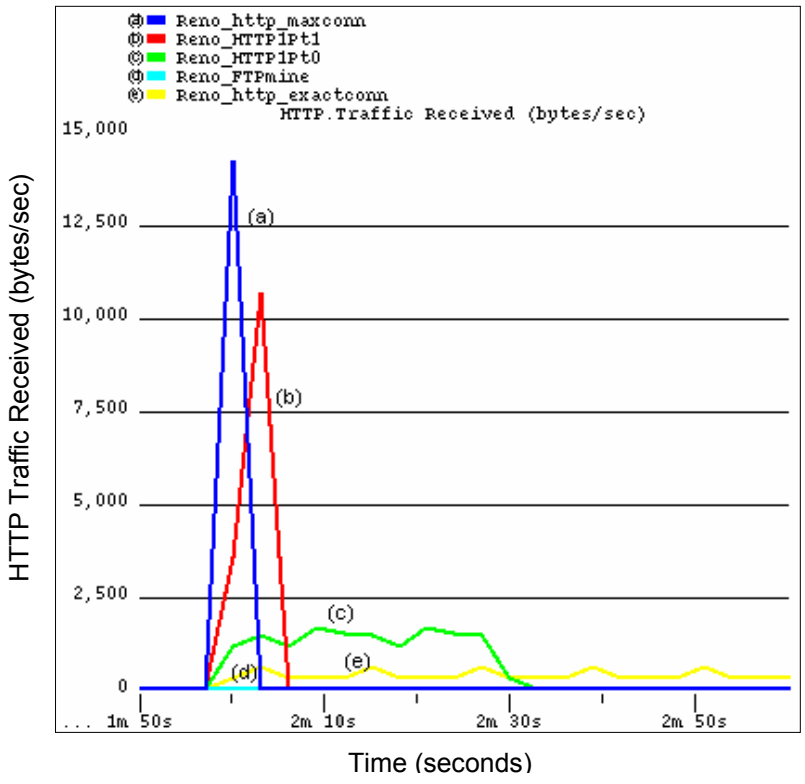

Figure 4.2: HTTP Traffic Received (bytes/sec)

This directly implies that the perceived latency for the user is lowest when using the MaxConn scenario followed by the HTTP/1.1 protocol. It is considerably high for FTP, HTTP/1.0 and ExactConn. Similarly, when we compared the scenarios with respect to the traffic sent, it was very similar for HTTP/1.1 and MaxConn (refer Figure 4.3).

We noted that the page response time, which is the time it takes for the entire base page to reach the client after the request was sent, was lower using MaxConn than with HTTP/1.1 and much lower than when HTTP/1.0 was used. Also, the object response time was as low or lower than HTTP /1.1 and much lower than HTTP/1.0 (refer Figure 4.4).

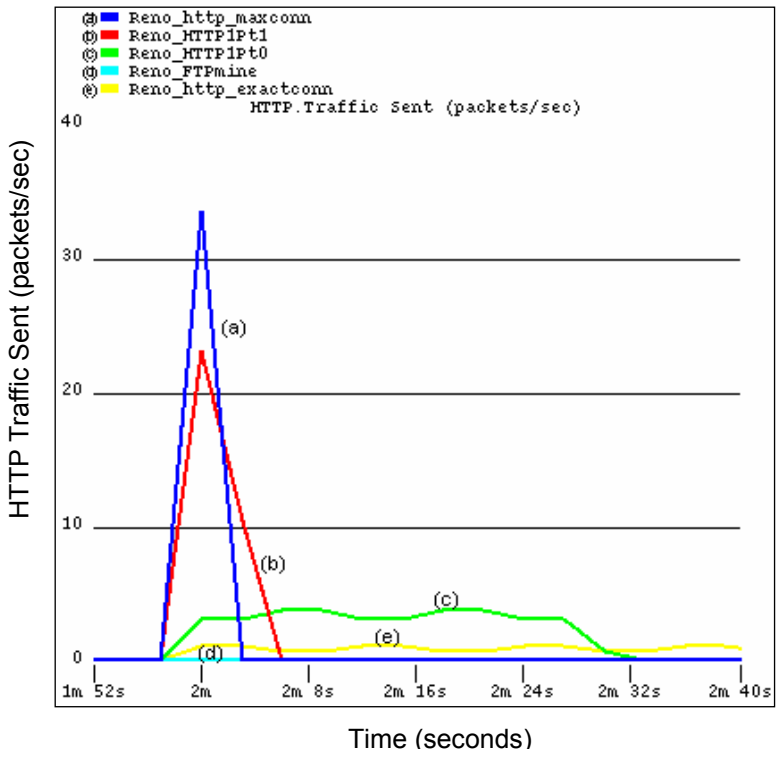

Figure 4.3: HTTP Traffic Sent (packets/sec)

The object response time is the metric that measures the time the objects embedded in the web page take to reach the client after the request was sent (refer Figure 4.5).

In the ExactConn scenario of the experiment, for the ESPN page, we noted that the traffic per second was moderately high, though over a longer time period. The page response time was slightly higher than with HTTP/1.1 and MaxConn but much lower than what it was with HTTP/1.0. Looking at the figures we observe that the performance of the MaxConn scheme is much better than that of HTTP/1.1. We hence concluded that multiple parallel connections work very well for a page with multiple elements, irrespective of their size.

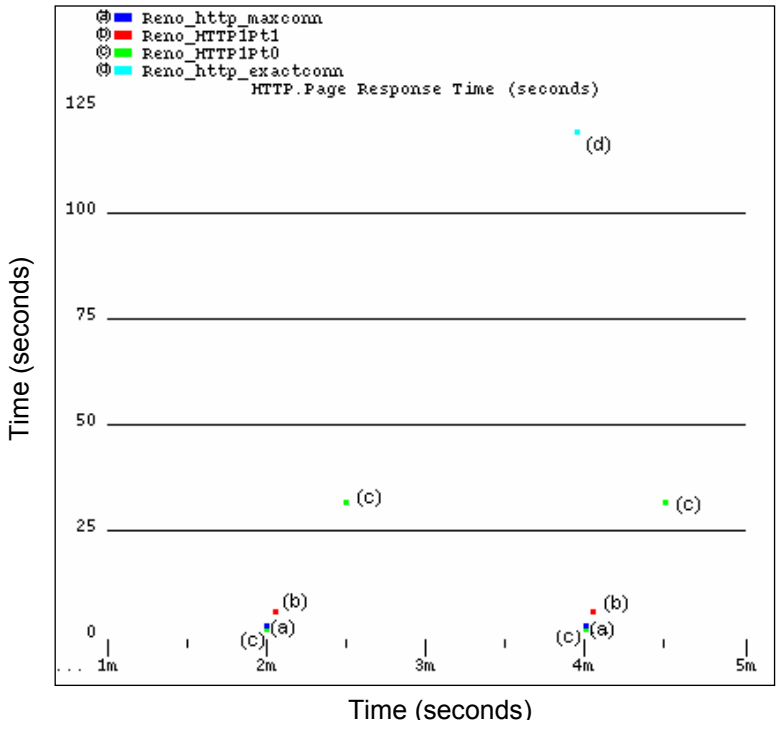

Figure 4.4: HTTP Page Response Time

We noted that in the case of a web page with many small or moderate elements, which is structurally very similar to the ESPN page that we had generated, the performance of the different scenarios in terms of the measured statistics such as number of connections spawned, traffic sent and received and object and page response times is very close to the performance of the respective scenarios while transferring the ESPN page. The traffic received per second using MaxConn was higher than with HTTP/1.1 and was much higher than with HTTP/1.0. Also, the page response time was lowest for MaxConn and was much better than all other scenarios including HTTP/1.1. The object response times were identical for HTTP/1.0 and ExactConn and are slightly higher for MaxConn and higher than that for HTTP/1.1, but the performance of MaxConn is still by far better as most of the object response times (if not all) overlap. This is completely unlike the HTTP/1.0 or the ExactConn scenarios which bring in objects, back-to-back.

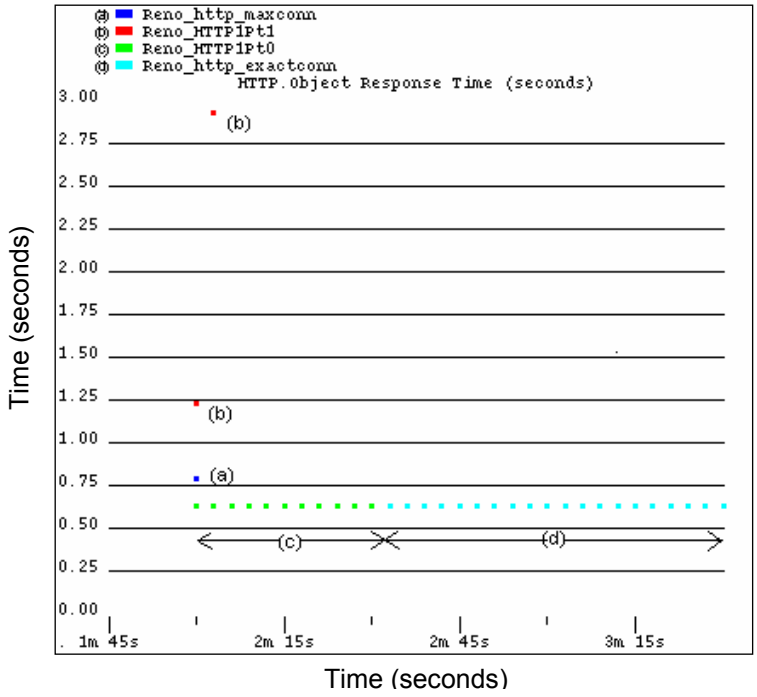

Figure 4.5: HTTP Object Response Time

We also ran experiments changing the number of elements on the page and observed that there was a definite and appreciable improvement in performance when a maximum number of connections were used. When we increased the number of connections from zero through the number of elements on the page, the performance in terms of lowered actual latency kept getting better and better as the number of connections approached the actual number of elements on the page. Identical results were observed even on increasing the number of connections above the number of elements on the page. Experiments run to transfer a page with different number of elements further validated these observations.

In the experiments, which involved transferring the page with one large element, we observed unexpected results. We noted that all the 4 application layer protocol variations had identical performances in all considered metrics. This obviously meant that HTTP/1.0, HTTP/1.1, MaxConn and ExactConn use the same connection to transfer the page and its contents. We hence concluded that increasing the initial cwnd does not contribute to performance enhancement substantially for any page irrespective of the nature of the web page i.e. whether the page has a small number of larger objects or a large number of relatively small or moderately sized objects.

4.5 Conclusions

By means of our various experiments using different generated flavors of HTTP we observed that the setting up of multiple parallel connections achieved the best results in terms of link utilization, transfer time, page and object response time, queuing delay and throughput. When we do not use connections that support pipelining, setting up many

connections is better than increasing the initial congestion window size of the connections. The performance of the system improves as the number of parallel connections approaches the number of embedded images on the web page. This is especially important since Netscape and IE browsers do not implement HTTP pipelining in spite of being HTTP/1.1 compliant. Thus we conclude that using multiple parallel non-persistent connections, without pipelining, leads to best overall results, in the current scenario, where web pages typically have several embedded objects.

We validated by running several experiments that opening up a single connection with a large CWND did not contribute to improvement in performance irrespective of the nature of the web-page. Hence, we did not consider exploring this scheme further.

4.6 A Critical Evaluation of the Adaptive Connections Scheme

There are several issues with the Adaptive Connections scheme that has been described in the sections above. Most importantly, multiple parallel TCP connections, lead to bursty traffic along the network links. Bursty traffic causes degradation in the performance of a TCP network. This is because traditional networks have intermittent routers in a link between the source and destination. These routers have queues of finite lengths. The generation of bursty traffic leads to the overflowing of these router queues which are typically drop-tail routers and this leads to packet loss [6][27]. This packet loss is immediately thought of as congestion by TCP, which drops its CWND to half the original size, as discussed earlier in this thesis. This leads to reduction of throughput, hence working against the aim of the multiple parallel connections scheme. Also, as the number of parallel HTTP over TCP connections increases and the packet loss rate

becomes higher due to congestion, the impact of multiple connections is decreased. The aggregate bandwidth will stop increasing or may even begin to decrease [28].

Another serious drawback of this scheme is that, setting up of multiple parallel connections between the server and the client, per individual client may lead to the server crashing because of the heavy server load. Typically a web-server serves several clients simultaneously. If each and every client uses the above algorithm, then the server will have to support an extremely large number of connections and will hence fail under the load.

Also, Ehsan et al claim, after extensive analysis that multiple parallel connections are not encouraged in traditional networks because it is a bandwidth hungry scheme even though it has shown to reduce latency considerably [22].

In addition, the implementation of this scheme assumes that the server is a "smart" server and it takes the responsibility of spawning the multiple parallel connections with the client. According to our algorithm, optimal performance is achieved when the number of simultaneous connections is close to or equal to the number of embedded objects in the web page. Hence smart servers are needed, because initially the client has no idea how many embedded objects the web page, it has requested, has. We note that the implementation of "smart servers" is a chimerical concept and may never happen.

An alternate solution to the above problem is that the client spawns the multiple parallel connections after it has received information from the server. This involves the client getting that information from the server before it sets the connections up. This would require an additional RTT, which beats the purpose of this scheme.

Another option for overcoming the above problem is the use of meta-information within the HTTP response headers. This approach is not very practical because it involves a change in the HTTP protocol. Hence we acknowledge that the setting up of multiple parallel connections is a formidable challenge.

4.7 Evolution of Adaptive Connections to Optimized Browser

In this section we look at how a modification to the Adaptive Connections scheme mentioned above, helps us overcome the issues involved with that scheme and achieve excellent performance in terms of user-perceived latency.

Traditionally, multiple parallel TCP connections have not been considered to be a good idea because of the adverse effect of TCP's Slow Start (SS) and Congestion Avoidance (CA) algorithms on the window size due to router queue packet drops caused by the generation of bursty traffic, as explained above. Also, this scheme is thought of as resource consuming. We note that in the scenario we are dealing with i.e. the HNS DIRECWAY/SPACEWAY we do not use TCP's SS and CA over the satellite link. Also, there are no intermittent routers between the NOC and the end user's antenna. This means that there is no risk of overflowing routers which will lead to TCP reducing its CWND size and hence dropping throughput due to the bursty nature of this scheme. We can hence say that the network under consideration here is not easily affected by bursty traffic.

Multiple parallel HTTP over TCP connections have shown to decrease user-perceived latency by 30% over HTTP/1.1 when used in transferring WWW content from

the server to the client via high delay links such as satellite links [28], in low and moderate traffic conditions.

Another point we take into consideration is that, most versions of Internet browsers do not implement HTTP/1.1 pipelining. Netscape/IE do not implement HTTP pipelining in spite of being HTTP / 1.1 compliant and use a LRU scheme/timeout to kill idle connections [8][9] as explained in section 2.2 of this thesis. Hence the reduction in latency using HTTP over multiple parallel TCP connections would be approximately 40-45% as compared to existing access methods and 30% over prototypes such as a HTTP Performance Enhancing Proxy (HPEP) [30]. These figures are true if a client could establish a number of connections with the server (NOC), equal to the number of objects on the web page.

We also note that since we are dealing with a satellite segment here, bandwidth on the space segment, though a resource, is not scarce in nature and can easily support "burst". This is an inherent characteristic of satellite resources. Therefore the issue raised in [24], mentioned in the previous section, does not have an impact in the network under consideration in this thesis because of its characteristic topology and bandwidth allocation characteristics.

The issue of requiring a smart server or the use of HTTP response header meta-information is solved by considering the topology of the network under consideration. We let the NOC be the "smart" element in this network. Thus it is the NOC that decides how many connections it can afford to establish with the client for individual requests at any given point in time. This is a very easy role for the NOC to play. Note that the NOC is connected to the WWW by a high-speed fiber connection which is the public Internet.

Hence the latency in this public portion of the network under consideration is negligible. Also, realize that the network between the NOC and the end-user, which involves the satellite link, is proprietary. Hence, if we reduce the latency in the proprietary part of the network, overall latency of the network is reduced. Because the high latency portion is proprietary, multiple parallel connections can be setup on this part without breaking the rules or requesting a change in the rules or protocols of the public Internet.

Also, since the NOC has already received the web-page from the web server, it knows the exact number of objects embedded in each and every page. Thus it is the NOC that has the responsibility of spawning multiple parallel connections with the end user. This solves the issue of the smart server. We call this scheme, where the NOC is the smart element, the Optimized Browser scheme. In this scheme the responsibility of spawning a number of multiple parallel connections from the NOC to the web-browser (client) for the purpose of retrieving the web resource from the NOC over the space segment with minimal delay, resides at the NOC.

Another advantage of this Optimized Browser scheme is the fact that since the NOC is aware of the congestion situation of the network, it can control it by deciding the maximum number of optimized connections to be allocated to respective clients, depending on metrics such as the time of the day, bandwidth utilization and the current and predicted (short-term) state of the network. Hence this scheme is scalable and dynamic. The NOC can also accomplish this by talking into consideration the user's service plan and give "gold" or "platinum" clients better service in terms of higher number of connections and hence reduced perceived latency.

4.8 Summary and Conclusion

We started this section with the motivation for the concept of adaptive number of connections and the development of the scheme in its infancy stages. We also described the details of the implementation and simulation experiments involving this scheme and showed that it has excellent performance in terms of latency in our network by means of our experiments and the interpretation of the results obtained. We then went on to discuss the possible issues with this scheme and why this scheme is not implemented in the current access methods for the WWW.

In the previous section we proved that the optimized browser scheme, in spite of the issues with traditional networks, is optimally suited for networks involving a satellite link and reduction in latency is guaranteed.

We hence conclude that the Optimized Browser scheme leads to a considerable reduction in actual latency leading to a 35-40% reduction in user-perceived latency, as shown in section 4.2, when used over networks involving a satellite link.

5 Proposed Scheme II – Cumulative Caching

In this section we focus on the cumulative caching scheme which tries to reduce the problem of high user-perceived latency using a different approach as compared to the optimized browser scheme (described in the previous chapter). The scheme relies on caching and does not modify the HTTP protocol in any way. This approach uses a change in the design of the network and the topology with minimal changes. The scheme has several advantages which include easy and inexpensive implementation and immediate savings in latency of up to 40%.

5.1 Introduction

This chapter is divided into six parts starting from the next section. In section 5.2, we briefly revise the definition of the problem we are trying to solve using this scheme by discussing the necessary background.

In section 5.3 we discuss the motivation for the cumulative cache scheme and then we go on to specify the algorithm it uses. We emphasize that we focus on a system that works with a single VSAT terminal supporting multiple users for Internet access i.e. Small Office Home Office (SOHO) setups.

In section 5.3, we discuss the motivation for the development of this scheme. We also specify the topology that is the target for the cumulative cache scheme. We continue in this section by elaborating the algorithm used by the scheme to paint web pages on the users browsers, i.e. the client processes for the first request and subsequent requests.

In section 5.4 we look at related work that has been done in this area. In the first part of this section we look at the work done in trying to analyze the nature of web-browsing and show via statistics that this scheme will indeed prove beneficial. In the latter part, we describe Zipf's law, its applications and its impact in this area.

In section 5.5, we go on to describe our observations and more importantly, we quantify the benefits of this scheme. Also, in section 5.6, our goal is to touch upon the implementation details of this scheme, where we mention some of the aspects that must be taken into consideration for commercial deployment.

Section 5.7 concludes this chapter and about the possible issues with the cumulative cache scheme and the future work in this area.

5.2 Background

HTTP is a request-response type application layer protocol [5] on TCP Reno. A HTTP transaction involves setting up a HTTP over TCP connection. This involves the traditional 3-way TCP handshake for connection establishment. The network we consider is the Hughes Network Systems (HNS) DIRECWAY® and SPACEWAY®. These are two-way direct to home broadband Internet systems. Two-way implies that both, the incoming data to the end user's terminal ("user") from the Internet and the outgoing data from the user to the web-server use the satellite segment. The basic network topology is as shown in Figure 1. Hence the Round-Trip Time (RTT) i.e. from the remote earth station to the satellite to the hub and from the hub to the satellite and back to the earth station can be larger than half a second as explained in Chapter 1 of this thesis.

This large RTT amplifies the effect of the TCP triple handshake which is required for a connection to be set up between a client and a server, say. Also, in TCP typically, the Maximum Segment Size (MSS) is 536 bytes. Client HTTP requests which are sometimes larger than the TCP segment size hence span multiple RTTs in addition to the initial connection setup overhead of HTTP over TCP to get through to the server [2]. TCP's Slow Start algorithm aggravates this delay as the second packet cannot be sent until the first has been acknowledged [31]. Hence the HTTP response may require multiple round trips to reach the server. This causes increased user-perceived latency due to HTTP transactions on this segment. This latency becomes apparent for the end-user when the transaction between the client and the server is actually a request from a computer to a web server and the response information is the base HTML web-page along with objects embedded in the base page. This makes "web-browsing" a cumbersome experience.

5.3 Motivation and Algorithm

The cumulative cache serves the entire group of end user machines served by one VSAT terminal for broadband access. The schematic of the network is as shown in Figure 1.1. Note that in the above figure, one VSAT terminal serves a group of end users i.e. human users on different computers, hence running different client programs (browsers). This setup is typical in a small office or a home office, where different users have common web browsing patterns, such as repeated hits to a web page that has data that is pertinent to the nature of their work. We hence onwards refer to this topology as the SOHO (Small-Office Home-Office) topology and customize our scheme especially to

benefit the users in this setup. Note that the cumulative cache is located between the users and the VSAT terminal. The cached content is typically application-layer content. The working of the cumulative cache is as simple as caching all the Internet content that passes through it.

This implies that all of the cacheable Internet content viewed by all users is cached in this cache. Since the number of users contributing to it is relatively large, it could be flushed every 24 hours (at lowest usage time of day) and it begins to refill the moment people start browsing in the morning. This way, the pages that are painted on the browsers are cumulatively cached and subsequent requests to those pages are cumulative cache hits. When the same client or another client requests the same page, the locally cached version will be displayed. At the same time the timestamp of this cached version will be sent over the satellite segment for validation from the cached version at the Network Operations Center (NOC). If the NOC has a cached version with a later time-stamp, it sends it over the satellite link to the client, where the client generates the new page and the browser auto-refreshes. Simultaneously, the NOC checks with the web-server and refreshes or upgrades the version it has cached. If the web-page cached at the NOC is upgraded, the NOC sends the newer page to the client over the satellite segment resulting in the client receiving the web-page and the browser auto-refreshing. The NOC refreshes/upgrades the page in its cache irrespective of whether the client has the same version as the NOC or otherwise. This ensures the freshness of the cached web resource at the NOC and hence obviates the second auto-refreshing of the client browser even in the worst-case scenario.

Note that this cumulative caching scheme is not the same as pre-fetching. It is a much simpler scheme. It does not use any fetching algorithm. The pages that have not been requested before are fetched from the source. This scheme also does not incorporate any fetching delay in the pages that have not been fetched before.

5.4 Related Work

In this section we discuss related work in this area. Specifically, we focus on work involving the nature of web browsing and Zipf's law's applications to web browsing.

5.4.1 Nature of Web Browsing

We now look into why this scheme will actually work and why it is especially beneficial for SOHO user networks. Caching in this environment is based on the assumptions that a large fraction of the HTTP responses have already been received and that these resources do not change between accesses. In [32] it is stated that 22% of the resources referenced in the traces they analyzed were accessed more than once. The first study in a related area, which used "live" users to test and see if the benefits would apply in practice, is given in [23]. In addition, this paper used two traces from independent sources for a trace-based analysis to quantify the potential benefits from both proxy-based and end-to-end applications. This paper claims that Internet browsers ("users") in the same geographic area visit the same websites due to the mirroring of certain web-servers or other reasons.

Also, users with the same nature of work visit the same websites according to this paper. The percentage of traffic, which is repeated by a single user was calculated in

terms of “delta-eligible” HTTP responses. Note that “delta-eligible” responses are ones, which reply with a different instance of the same resource (HTTP Status code 200). In the traces, 20-30% of the status 200 HTTP responses were delta-eligible i.e. changed slightly from what was cached. Even in the status 200 HTTP responses, 30 % were identical to what was cached. This paper ignored the responses that had the same instance of the resource as the one that was cached (HTTP Status code 304). In spite of trying to filter out these “not-modified-since” responses, that number was 14% of the total number of responses in the trace.

More recent studies show that 15-18% [33], 30% [34] and 37% [35] of HTTP requests responded with Status Code 304, i.e. cached copy is up-to-date. Also, in [36] it is stated that it is well known that 20% to 30% of all requests are conditional GET requests with 304 (not modified) replies. This means that for an individual user, between 15% and 37% of all responses are repeated. If we were to use the cache as a cumulative cache for a group of users in the same geographical area and with the same nature of work, we achieve at least 40% and up to 100% hits in the cache.

The usefulness and promise of the cumulative caching scheme is validated by these statistics.

5.4.2 Zipf's law

We also look into some recent work which deals with the application of Zipf's law to the nature of web-browsing. Zipf's law states, "The probability of occurrence of words or other items starts high and tapers off. Thus, a few occur very often while many others occur rarely." This theory when applied to web access has been claimed to be equivalent to the fact that, user visits a certain small percentage of web resources often and visits a large number of other web-pages very infrequently. This means that on the average, say, 90% of all HTTP requests by a web browser are directed towards only 10% of the online resources it accesses and the remaining 10% of the HTTP requests are for the remaining 90% web resources.

This is very interesting for our scheme, because it means that even if the cumulative cache saves only 10% of all the web resources that pass through it, most users will benefit 90% of the time i.e. the perceived network latency will not appear 90% of the time.

A study by Pei Cao gives us numbers that validate the fact that even though the figures are not as mentioned above, (i.e. 90% of accesses go to 10% of items), they are very close to the "90/10" rule, even for web accesses seen by a proxy. For example, 25% of all documents accounts for 70% of Web accesses in DEC, Pisa and FuNet traces, while it takes 40% of all document to draw 70% of Web accesses in UCB, QuestNet and NLANR [36]. Hence, realistic figures for a Zipf-like distribution for web requests are 70/30.

A paper by Breslau et al addresses two similar issues. The first issue is whether Web requests from a fixed user community are distributed according to Zipf's law and the

second issue is whether this characteristic is inherent to web accesses or not. On investigating the page request distribution, the paper shows that the answers to the two questions are related [37]. The paper also conforms with the results of [36] in finding that the page request distribution does not follow Zipf's law precisely, but instead follows a Zipf-like distribution with the exponent varying from trace to trace. They considered a simple model where the Web accesses are independent and the reference probability of the documents follows a Zipf-like distribution and found that the model yields asymptotic behaviors that are consistent with the experimental observations. This suggests that the various observed properties of hit ratios and temporal locality are indeed inherent to Web accesses observed by proxies.

5.5 Observations and Benefits

In this section, we quantify the benefits of the cumulative cache. The cumulative cache scheme is very well supported by the observations made in the papers quoted above. We now calculate the response times for HTTP responses from different points in the network. We assume a large page size i.e. 50kB (base) + 100kB (sum of all embedded object sizes in the web page). Hence the total page size is 1200kb. The time to transfer the page from the browser cache (RAM) is about 50 ns access time and file transfer at the rate 200 Mbps. This becomes approximately 6 ms which is perceived as instantaneous by the user. Similarly, the time required to transfer the same web-page from the cumulative cache (Ethernet LAN at a transfer rate of 100 Mbps) is a little over 12 ms. This too is perceived as instantaneous by the end user i.e. no perceived latency.

The time to transfer the same page from over the satellite segment i.e. from the NOC is calculated as follows. SPACEWAY® data rate at upload will be as high as 16Mbps and for downloads as high as 50Mbps. In this case, we ignore the 3-way TCP handshake. We assume HTTP/1.1 compatible and p-HTTP compatible network. Hence we consider just the NOC search time (RAM access) plus request transfer time (0.5 sec) + response transfer time (0.5 sec) + page transfer time (25 m seconds), which is about a little over 1 second. This delay is aggravated by the request-response behavior of HTTP as discussed earlier in this thesis. Note that if a new HTTP over TCP connection has to be set up we add another 1.5 second of the TCP triple handshake for connection establishment. Hence we have 2.5 seconds of actual latency which contributes to the increased perceived latency because of HTTP behavior. Now the time range required to transfer the web-page from the web-server (Internet) is similarly calculated to be between 3 seconds and 4.5 seconds.

Note that, the reduction in latency by 70% repetition of requests for web resources by an individual is 40%. Out of the remaining 60%, up to 42% could overlap with other users browsing patterns in the SOHO network. We define “cumulative resources” as resources that have been or will be requested by at least one other user in the SOHO network. Hence the probability of requesting a cumulative resource is 0.7 by the 70/30 variant of Zipf’s law. “N” users in the SOHO network are equally likely to request a cumulative resource. Hence, the probability of any individual user requesting a cumulative resource is 0.7/N. This implies that the probability of any individual user not requesting a cumulative resource first is 1-0.7/N. Note that not requesting a cumulative resource first, implies that some other user in the SOHO group has requested it earlier.

This implies a 100% reduction in the latency for the arrival of this resource at the client. If the number of users in the SOHO network is 10, i.e. N = 10, which is a very realistic figure, the reduction in latency in the remaining 60% is equal to the probability of not requesting a cumulative resource first, which is 93%. This translates into an additional reduction of up to 39% in addition to the 40% reduction in latency due to the self-repetition nature of web requests of the individual user. This amounts to a total reduction in latency of up to 79% using the cumulative cache.

These figures show explicitly the benefit of cumulative caching. This implies a 40% through 79% reduction in the user-perceived latency in direct proportion to the hit-ratio of the user to the cumulative cache. Following the discussing on Zipf's law, savings close to this can be achieved even if all of the Internet content passing through the cache is not saved and some kind of "smart" caching scheme is employed, which caches only the most requested responses.

The cumulative cache also eliminates the request and response traversing the satellite segment, thereby reducing the traffic on the space segment of the network. Even though we validate the cached versions in the cumulative cache and the NOC using time-stamps, the effective bandwidth utilization is up to 20 times lower than otherwise.

The cumulative cache scheme is also very useful in cases when a surfer arrives at a web page expecting or knowing that this will lead him/her to another. This current page acts like a portal to another whose address he/she does not know. The performance of our scheme is excellent in such cases as here the freshness of the former page is not important so long as it has the link or other information that the transient user desires.

5.6 Implementation

We summarize the implementation details of the cumulative cache in this section. Please note that we do not detail upon what cache replacement algorithms to use and assume a non-realistic but simplistic approach that the benefits we obtain are directly proportional to the cache size.

Currently HNS uses a set-top box which runs at the network layer, as part of the IDU (Indoor Unit) with the DIRECWAY® system at the SOHO VSAT terminal. This "box" needs to be provided with additional memory and enabling its working at the application layer will make it function as the cumulative cache. This might prove to be expensive for the service provider as the equipment cost increases with every kilobyte of storage. Instead of using additional memory provided by the service provider, the end user (SOHO network) could be encouraged to use a fraction of the hard-disk space available for the cluster cache to curb additional aggregation to product cost. Due to this, this scheme can be implemented by the service provider, HNS in our case, as optional but recommended. Since SPACEWAY® focuses on the SOHO market, the incoming traffic could be directed through one of the user's computers where a fraction of the (or an entire) hard disk could be configured to cache incoming WWW data, using a daemon process.

We may keep this process transparent to the end user or may let the end user know by displaying a "Page is being Verified" sign or equivalent while the cached page is being displayed and confirmation about its freshness has not been received from the source i.e. the web server. The risk involved in displaying outdated web-pages, for a few seconds, which is instant gratification for the end user is lightened by the fact that most

web designers change just some form or appearance of the web-page but not the content of the page in order to give the webpage a fresh look [38]. We also note that the probability of an outdated page being displayed to the user is miniscule in the cumulative cache scenario, as the cache may be flushed every 24 hours and it begins to fill at the start of business, everyday.

5.7 Conclusions and Future Work

By means of our discussion above we observe that the setting up of a cumulative cache for a SOHO-type network achieves very good results in terms of the user's perceived latency for WWW access using a satellite link. Our observations suggest a minimum reduction of 40% and up to 100% in the user's perceived latency, using this scheme. The application of Zipf's law to this scenario shows that very similar results can be achieved by caching a much smaller number of HTTP responses if a smart caching scheme, which caches only the most requested web-pages is developed.

We are currently analyzing the risk of displaying outdated pages even momentarily to the user and the effect that he/she has knowledge of the same has on his/her view on satisfactory web-browsing. We are also looking into the issue of privacy and making sure that no individual user has access to the contents of the cumulative cache as it may contain sensitive information such as personal or financial information of other users. Hence the cumulative cache must be securely protected against unauthorized access and must be used for sharing non sensitive information i.e. the cache should be accessible only by the browser process and this should be transparent to the user.

We also plan to look into the cost/benefit ratio – size of cache (cost) to perceived reduction in latency (benefit) – to determine the optimal size of the cumulative cache for a fixed number of users in the SOHO network along with the development of an appropriate cache replacement algorithm.

6 Conclusions

We have studied and discussed the optimized browser and cumulative cache schemes in the previous two chapters. In this chapter we briefly describe out conclusions regarding the pros and cons of these schemes.

6.1 Optimized Browser

Observations made in chapter 4 show that the setting up of multiple parallel connections achieved the best results. The metrics that we observed were link utilization, transfer time, page and object response time, queuing delay and throughput. We also observed that if we do not use connections that support pipelining, setting up many connections is better than increasing the initial congestion window size of the connections. The performance of the system keeps increasing as the number of parallel connections approaches the number of embedded images on the web page.

Since Netscape and IE browsers, which are the two most widely used browsers, do not implement HTTP pipelining in spite of being HTTP/1.1 compliant, we conclude that using multiple parallel non-persistent connections, without pipelining, leads to best overall results. This conclusion is based on the assumption that the web resource that we wish to transfer typically has several embedded objects. This assumption is more of a rule than an exception with the advent of broadband Internet connections, Macromedia Flash technology and different compression algorithms for images, graphics and streaming audio and video content.

We observed that opening up a single connection with a large CWND did not contribute to improvement in performance irrespective of the nature of the web-page.

We also conclude that the spawning of multiple parallel connections will contribute to the reduction of the user's perceived latency while browsing the WWW in the scenario we are dealing with i.e. the HNS DIRECWAY/SPACEWAY where we do not use TCP's SS and CA over the satellite link. Also, there are no intermittent routers between the NOC and the end user's antenna. Hence there is no risk of overflowing routers which will lead to TCP reducing its CWND size and hence dropping throughput due to the bursty traffic produced by this scheme. We conclude that the network under consideration here is not easily affected by bursty traffic in a derogatory way.

The reduction in latency using HTTP over multiple parallel TCP connections would be approximately 40-45% as compared to existing access methods, since most popular browsers do not implement pipelining and 30% over prototypes such as a HTTP Performance Enhancing Proxy (HPEP).

Since the NOC would be the "smart" element in this network, which decides the number of connections allocated between any client and itself, there is no need for the implementation of smart servers. This leads to the reduction of latency in the proprietary part of the network, and as a result overall latency of the network is reduced. The NOC knows the exact number of objects embedded in each and every page and also the state of the network with respect to traffic and congestion. Hence the implementation of the Optimized Browser scheme, which is scalable and dynamic, becomes a straight-forward matter.

6.2 Cumulative Cache

In chapter 5, we discussed the benefits of the cumulative cache. We conclude that using the cumulative cache for a group of users in the same geographical area and with the same nature of work i.e. a SOHO environment, could achieve at least 40% and up to 100% hits in the cache per individual user.

Cumulative caching implies a 40% through 79% reduction in the user-perceived latency in direct proportion to the hit-ratio of the user to the cache. Following the discussion on Zipf's law, savings close to this can be achieved even if all of the Internet content passing through the cache is not saved and some kind of "smart" caching scheme is employed, which caches only the most requested responses. The cumulative cache eliminates the request and response traversing the satellite segment, thereby reducing the traffic on the space segment of the network. The effective bandwidth utilization is up to 20 times lower than otherwise.

We also conclude that this caching scheme is very useful in cases when a surfer arrives at a transient web page expecting or knowing that this will lead him to another resource which is his final destination for the moment, at least.

The existing equipment at the end user's SOHO network can be used in this setup and this scheme can be hence implemented as optional but recommended. The risk involved in displaying outdated web-pages, for a few seconds, which is instant gratification for the end user is lightened by the fact that most web designers change just some form or appearance of the web-page but not the content of the page in order to give the webpage a fresh look. We also conclude that the probability of an outdated page

being displayed to the user is miniscule in the cumulative cache scenario, as the cache may be flushed every 24 hours and it begins to fill at the start of business, everyday.

By means of our discussion we conclude that the setting up of a cumulative cache for a SOHO-type network achieves very good results in terms of the user's perceived latency for WWW access using a satellite link. The application of Zipf's law to this caching scheme shows that significant reduction in latency can be achieved by caching a much smaller number of HTTP responses if a smart caching scheme, which caches only the most requested web-pages is developed.

Thus we conclude that the cumulative cache is a very useful scheme for SOHO-type network topology at the client end. Reduction of 40% through 79% in perceived latency can be obtained using this scheme.

7 Future Work and Other Schemes

This chapter is divided into two parts. The first part deals with the future work and possible resolution of the issues with the two schemes introduced in this thesis; in chapters 4 and 5. The second part of this chapter deals with some schemes that can be developed to reduce the users' perceived latency when dealing with WWW transfers over high delay links such as GEO satellite links. These schemes have been suggested and worked upon in the past as can be seen from the references provided. We have introduced some new variations in these schemes which we feel will benefit the performance of the schemes especially in our scenario. The primary difference between the schemes described in these sections and the schemes described in chapters 4 and 5, is that, the schemes in this chapter have not reached a level of maturity that is present in the others. Nevertheless, we describe them briefly in this chapter, for the sake of completeness and since these may define future work in this area.

7.1 Future Work

This section is divided into two sub sections. In sub-section 7.1.1 we discuss the adaptive connections scheme presented in chapter 4. We primarily address the unresolved issues with this scheme and possible future work in the area. Similarly, in section 7.1.2, we discuss the cumulative cache scheme, presented in chapter 5 of this thesis. Again, we address unresolved issues and future work.

7.1.1 Optimized Browser Scheme

The optimized browser scheme that we have described in this thesis does not specify the number of connections to be setup by the NOC with the client in case the usage of the space segment of the network or overall throughput of the system is high. Also, we need to find out the number of connections that can be spawned by the NOC, in case the number of embedded objects on the web resource is extremely large. A tradeoff can be established between the number of concurrent connections and the pipelining of objects on these connections. Thus a maximum and minimum number of connections needs to be pre-decided. This number has to be independent of the nature of the requested web page, in terms of the number of embedded objects in the page. We will also look into finding an optimal number of connections to be set up depending on the number of elements in the web page.

Future work in this area also consists of, but is not limited to studying the effect of multiple parallel connections on the processing load at the NOC. The effects of this scheme on the bursty bandwidth utilization over the space segment also need to be analyzed.

7.1.2 Cumulative Cache Scheme

The cumulative cache scheme displays the web resource on the user's browser, even before the freshness of the resource has been validated by the response that arrives from the source. We are currently analyzing the risk of displaying outdated pages even momentarily to the user and the effect that he/she has knowledge of the same has on his/her view on satisfactory web-browsing. The issue of whether this transaction

should/could be transparent to the user and its legal implications also constitute future work in this area.

As of now the cache flushing period is set to 24 hours. This is based on studies that show that the average frequency of changing the content or structure of web pages is much higher than one day. We realize that the higher the time period between cache flushes, the greater the perceived latency and bandwidth utilization benefit. Finding the optimal time period between cache flushes is an issue that needs to be addressed.

We are also looking into the issue of privacy and making sure that no individual user has access to the contents of the cumulative cache as it may contain sensitive information such as personal or financial information of other users. Hence the cumulative cache must be securely protected against unauthorized access and must be used for sharing non sensitive information i.e. the cache should be accessible only by the browser process and this should be transparent to the user.

The cost/benefit ratio of the cumulative cache, i.e. the size of cache (cost) to perceived reduction in latency (benefit) needs to be looked into. This will help us determine the optimal size of the cumulative cache for a fixed number of users in the SOHO network. The development of an appropriate cache replacement algorithm, specifically targeted towards the cumulative cache is also an important needed study in this area.

7.2 Other Schemes

In this section of this thesis we very briefly, discuss other improvements on existing schemes or completely different schemes which we believe can greatly assist in

the reduction of the user perceived latency in a network similar to the one under consideration here. Note that these schemes are not yet mature and this section can be thought of as one that defines the additional future work in this area.

The first scheme in sub-section 7.2.1, deals with the "fast delta" algorithm. In this scheme, the entire delta of the web resource is not generated. Only the deltas of specific parts of the web page, such as the text portion, are generated. This scheme is useful while transferring web pages with pre-compressed data such as images, audio or video. Sub-section 7.2.2 deals with the pre-creating of deltas and pre-fetching of web resources. The success of this scheme depends on the statistics obtained from real Internet traffic traces.

7.2.1 The "fast" delta algorithm

This algorithm takes off from where Banga et al [39] leave their optimistic delta algorithm. In the optimistic delta algorithm, deltas generated from possibly outdated web resources may be transferred from the server to the client followed up by a subsequent delta, which updates that information. This scheme is called "optimistic" because it assumes that there will be enough idle time to transfer most of the older version of the web resource before the newer version in available. Also, the optimistic delta scheme assumes that the deltas, as explained in chapter 2 of this thesis, are smaller than the actual web resource.

The fast delta algorithm does not change or discuss the algorithm which is used for the creation of deltas as is discussed in [40], or quantify the benefits of delta encoding used for HTTP transactions as is done in [23], [24] and [39]. Our scheme modifies the algorithm in which delta encoding is implemented for HTTP i.e. WWW transfers over

high delay links. Note that the optimistic delta scheme generates the delta of the older version of the web resource and sends it over to the client for regeneration before it sends the newer version of the resource. It is also interesting to note that all the responses originate from the web server and not from any intermediate proxy or cache. This optimistic delta scheme, unlike any traditional delta scheme, results in possible increase in overall network traffic.

The "fast" delta algorithm uses the most efficient vdelta / vcdiff [40] difference generator to create the deltas of only the text portions of web pages. Our algorithm does not calculate the deltas of non-text parts or pre-compressed parts of the web resource. This means that the deltas of pseudo-text objects (word processors, spreadsheets, etc.) and object files are not generated. We also do not consider calculating the deltas of multi-dimensional objects such as images, audio or video.

We feel that this scheme can have a huge impact on web pages such as news pages where the structure of the page is retained and only the content i.e. text changes varies frequently. These types of web pages constitute a significant portion of web resources as pointed out in [32]. Due to the fact that the deltas of only one-dimensional objects are created, it becomes very easy to create and cache the delta of non-dynamic web pages and the templates of dynamic web pages. The creating, caching, retrieval and validation of these deltas of different instances can be done efficiently using the <TEMPLATE> and <DYNAMIC> HTML tags as suggested in [41].

The biggest advantages of this scheme are the significant reduction in the size of the resource or traditional delta that needs to be transferred and the fact that this algorithm is faster than the one proposed by Banga et al in [39] as the regeneration of

text-based deltas is almost instantaneous. The scheme is also useful for pages where the text content of the web resource is high.

The drawbacks of this fast delta scheme is that its usefulness is restricted to a certain subsection of the web resources online.

7.2.2 Pre-creating and Pre-fetching

Last but definitely not the least, this subsection deals with the merger of two very promising schemes, i.e. the pre-creating of deltas and the pre-fetching of web resources. Pre-creating of deltas means the creation and storage of a delta of a web resource, every time it is updated or changed. The deltas are created and stored in anticipation of a request from a client which has cached an earlier version of the web resource. These deltas are then sent by the source server to intermediate proxies and caches. At these intermediate proxies and caches the newer version of the resource could be generated using the newly arrived delta and the older cached resource and the delta cached by itself as well. This way, this proxy or cache can provide the latest version of the web resource to completely new clients who request it for the first time and also send only the delta to clients which have the older version of the delta cached.

Note that due to the above “propagation” of deltas, the reduction in latency is not only the reduction in the processing time involved in the creation of the delta, but also the time involved in transferring the delta from the source to the intermediate proxy or cache. This may very well be the segment of the network involving the high delay link.

The NOC and the other intermediate proxies could also be involved in the pre-creation and caching of the deltas. This will require some kind of a multicast algorithm in order to propagate the deltas efficiently.

The biggest drawback of this scheme, as may have become obvious from the description above, is the increased network traffic which may lead to serious congestion issues. This issue can be lightened by making sure that the creation and propagation of deltas occurs at times when the server and network are least loaded respectively. Note that, since the number of times the user requesting a certain page may be different for different pages, we need to establish a threshold, depending on the page hits, below which pre-creating of deltas does not have a high enough cost/benefit ratio. This scheme also suffers from the other drawbacks of all delta encoding schemes such as cache bursting and others described in chapter 3 of this thesis and the ones described in [42]. This is an IETF memo that discusses problems with proxies that specifically act as intermediaries for WWW requests and more specifically caching proxies, which retain copies of previously requested resources which aim at reducing the overall latency by serving the requested web content, locally. The issues this document discusses are the possible loss of client cache directives due to proxies, the hindrance caused by intermediate proxies towards implementing new HTTP methods, loss of authentication due to intermediate proxies, bandwidth utilization issues that are caused by several proxies responding to one request and the freshness of content. This document also addresses some implementation issues.

Pre-fetching of the web resource, on the other hand employs a smart client algorithm in which the client machine can predict what resource the user will request next

as is described in [43]. In other words, the client machine predicts what hyperlink the human user will click on next and asks for that resource before the user has even requested it. In this case, the request-response transaction occurs when the user is looking at the web page displayed to him on his browser and finally when he requests the predicted web page; it is already present in his cache for the client machine to display almost instantaneously.

We feel that depending on the statistics obtained from real-life traces of Internet requests, of the most popular web-pages, we can often predict to a highly accurate degree what web-page a client will request next. On receiving the expected pages, while the user is looking at the web page painted on his browser, the client could cache the page (if they have not been received earlier) or cache the delta (if the same page has been requested earlier). Here we propose the blend of the usage of pre-created deltas and predictive pre-fetching as we expect that this could lead to an almost 100% reduction in the users perceived latency, depending on the hit-ratio of the pre-fetching algorithm.

We expect that this scheme will, not only reduce the absolute latency because of delta encoding but will further reduce the perceived latency by the client because of pre-fetching of the deltas and regeneration of the page at the client. We further anticipate a scheme which will let the server differentiate between requests for the anticipated deltas and those directly by the human user and hence giving the latter higher priority.

The implementation of this scheme could be done via a process at the client side which requests future pages (knowing the "browsing profile" of its user) and another process at the server side which sends the requested data to the client. The process at the server side needs to differentiate between requests that are being made for the future and

requests that are made directly by the user and give them higher priority. Another possibility is to let the NOC take up the responsibility of remembering the client, predicting the user's behavior and transmitting the pre-created deltas to the client. This takes away the processing load on the user's machine.

The blend of the two schemes mentioned above shows great promise in the reduction of absolute latency by reduction in the number of bits transferred over the high delay link and also the reduction of perceived latency of the user by predicting the pages the user will request shortly and by requesting them before hand. The drawback of this scheme is the lack of an accurate predictive algorithm which improves the overall performance of the system without increasing the network utilization to an extent of causing congestion in the system.

BIBLIOGRAPHY

[1] M. Allman, V. Paxson and W. Stevens, “TCP Congestion Control”, IETF-RFC 2581, April 1999.

[2] Simon Spero “Analysis of HTTP Performance Problems” Available at: http://metalab.unc.edu/mdma-release/http-prob.html, July 1994

[3] T. Berners-Lee, R. Fielding and H. Frystyk, “Hypertext Transfer Protocol -- HTTP/1.0”, IETF-RFC 1945, May 1996

[4] Joe Touch, John Heidemann and Katia Obraczka "Analysis of HTTP Performance" Information Sciences Institute, University of Southern California, Available at: http://www.isi.edu/lsam/publications/http-perf/, August 1996.

[5] R. Fielding, J. Gettys and J. Mogul, "Hypertext Transfer Protocol -- HTTP/1.1", IETF-RFC 2616, June 1999

[6] Mozilla Network Library Documentation, Available at: http://www.mozilla.org/projects/netlib/, September 2003

[7] Balachander Krishnamurthy, Jeffrry Mogul, David Kristol, "Key Differences between HTTP/1.0 and HTTP/1.1", Available at: http://www.research.att.com/~bala/papers/h0vh1.ps.gz, December 1998

[8] Zhe Wang and Pei Cao, "Persistent Connection Behavior of Popular Browsers", Department of Computer Sciences, University of Wisconsin, Madison, Available at: http://www.cs.wisc.edu/~cao/papers/persistent-connection.html, 1998

[9] Venkata Padmanabhan and Jeffrey Mogul "Improving HTTP Latency", In the proceedings of the 2nd World Wide Web Conference '94: Mosaic and the Web, pages 995 -1005, October 1994

[10] H. Nielsen, P. Leach, S. Lawrence, "An HTTP Extension Framework”, IETF-RFC 2774, February 2000

[11] K. Moore, N. Freed, "Use of HTTP State Management", IETF-RFC 2964, October 2000

[12] John Heidemann, Katia Obraczka, Joe Touch "Modeling the Performance of HTTP over Several Transport Protocols", ACM/IEEE Transactions on Networking 5, pages 616-630, October 1997

[13] D. Eastlake, C. Smith and D. Soroka, “HTTP MIME Type Handler Detection”, IETF-RFC 2936, September 2000

[14] Henrik Nielsen, Jin Gettys, Anselm Baird-Smith "Network Performance Effects of HTTP/1.1, CSS1 and PNG", In the Proceedings of ACM SIGCOMM `97, Cannes, France, September 1997

[15] The GZIP Home Page, “http://www.gzip.org”

[16] Timothy J. McLaughlin, “The Benefits and Drawbacks of HTTP Compression”, Leigh University, Available at http://www.cse.lehigh.edu/techreports/2002/LU-CSE-02-002.pdf, February 2003.

[17] The GZIP algorithm, http://www.gzip.org/algorithm.txt

[18] J. A. Thomas and T. M. Cover, “Elements of Information Theory”, Wiley, 1991

[19] Mark Stacy "Suggested Performance Improvements for HTTP" 1998

[20] Hans Kruse, Mark Allman, Jim Griner and Diepchi Tran "Experimentation and Modeling of HTTP over Satellite Channels" International Journal of Satellite Communications, 19(1), January/February 2001

[21] M. Allman, S. Floyd, C. Partridge, "Increasing TCP's Initial Window", IETF-RFC 2414, September 1998

[22] Navid Ehsan, Mingyan Liu, Rod Ragland, "Measurement Based Performance Analysis of Internet over Satellites", 2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2002), San Diego, July 2002

[23] Jeffrey Mogul, Fred Douglis, Anja Feldmann, "Potential Benefits of Delta Encoding and Data Compression for HTTP" in the Proceedings of SIGCOMM '97, September 1997

[24] J. Mogul, B. Krishnamurthy, A. Feldmann, "Delta encoding in HTTP", IETF-RFC 3229, January 2002

[25] Xiaoming Zhou and John S. Baras, “Congestion Management in Access Networks with Long Propagation Delay”, ISR Technical Report (TR 2003-46), Center for Satellite and Hybrid Communication Networks, University of Maryland, October 2003.

[26] Xiaoming Zhou and John S. Baras, "TCP over satellite hybrid networks", ISR Technical Report (TR 2002-27), Center for Satellite and Hybrid Communication Networks, University of Maryland, October 2003, February 2002.

[27] Shawn Ostermann, Mark Allman and Hans Kruse, "An Application-Level Solution to TCP's Satellite Inefficiencies" WOSBIS 1996

[28] Thomas J. Hacker, Brian D. Athey and Brian Noble, "The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network", In the proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), Fort Lauderdale, April 2002

[29] Aniruddha Bhalekar and John S. Baras, "The Experimentation and Modeling of Satellite-friendly HTTP", In the Proceedings of the 37th Annual Conference on Information Sciences and Systems, Johns Hopkins University, Baltimore, March 2003

[30] HNS HTTP Performance Enhancing Proxy (HPEP) Specifications, Available at: http://www.hns.com/pdfs/DW_HPEP_LR.pdf

[31] M. Allman, V. Paxson and W. Stevens, "TCP Congestion Control", IETF RFC 2581, April 1999

[32] Fred Douglis, Anja Feldmann and Balachander Krishnamurthy, "Rate of Change and other Metrics: a Live Study of the World Wide Web", in the Proceedings of USENIX Symposium on Internetworking Technologies and Systems, December 1997.

[33] Balachander Krishnamurthy, Craig E. Willis, "Piggyback Server Invalidation for Proxy Cache Coherency", In the proceedings of the 7th International WWW Conference, Pages 185-193, Brisbane, Australia, 1998.

[34] Eric Nahum, "WWW Workload Characterization work at IBM Research", World Wide Web Consortium Web Characterization Workshop, Cambridge, MA, November 1998.

[35] Martin Arlitt and Tai Jin, Workload Characterization of the 1998 World Cup Website, Technical Report HPL-1999-35, HP Laboratories, Palo Alto, CA, 1999.

[36] Pei Cao, "Characterization of Web Proxy Traffic and Wisconsin Proxy Benchmark 2.0", Position Paper in World Wide Web Consortium Workshop on Web Characterization, Cambridge, MA, November 1998.

[37] Lee Breslau, Pei Cao, Li Fan, Graham Phillips and Scott Shenker, “Web Caching and Zipf-like Distributions: Evidence and Implications”, in the proceedings of the Conference on Computer Communications, IEEE INFOCOM, New York, March 1999.

[38] Luis Francisco-Revilla, Frank M. Shipman III, Richard Furuta, Unmil Karadkar and Avital Arora, “Perception of Content, Structure and Presentation Changes in Web-based Hypertext”, in the proceedings of the 12th ACM conference on Hypertext and Hypermedia, Arhus, Denmark, 2001.

[39] Gaurav Banga, Fred Douglis and Michael Rabinovich, “Optimistic Deltas for WWW Latency Reduction”, USENIX Technical Conference, 1997

[40] James Hunt, Kiem-Phong Vo and Walter Tichy, “Delta Algorithms: An Empirical Analysis” ICSE SCM-6 Workshop, 1996

[41] Stephen Williams, “HTTP: Delta Encoding”, Virginia Polytechnic Institute and State University, Available at: http://ei.cs.vt.edu/~williams/DIFF/prelim.html, 1997

[42] I. Cooper, J. Dilley, "Known HTTP Proxy/Caching Problems" IETF-RFC 3143, June 2001

[43] Venkata Padmanabhan and Jeffrey Mogul, "Using Predictive Pre-fetching to Improve World Wide Web Latency", In the Proceedings of the ACM SIGCOMM '96 Conference, 1996