A SIMULATION STUDY ON HTTP PERFORMANCE ANALYSIS IN TERMS OF ITS INTERACTION WITH TCP

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCES
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Deniz Gürkan
September 1998

A SIMULATION STUDY ON HTTP PERFORMANCE ANALYSIS IN TERMS OF ITS INTERACTION WITH TCP

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS

ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Deniz Gürkan

Deniz Gürkan

September 1998

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Erdal Arıkan(Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Ezhan Karaşan

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Murat Alanyalı

Approved for the Institute of Engineering and Sciences:

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

A SIMULATION STUDY ON HTTP PERFORMANCE ANALYSIS IN TERMS OF ITS INTERACTION WITH TCP

Deniz Gürkan
M.S. in Electrical and Electronics Engineering
Supervisor: Prof. Dr. Erdal Arıkan
September 1998

In this thesis, we have performed a simulation study on performance analysis of HTTP (HyperText Transfer Protocol) in terms of its interaction with TCP (Transmission Control Protocol). The latency through internet connections can be reduced by making modifications on the application and transport layer protocols. For the simulations, we have built models of HTTP/1.0 and HTTP/1.1 using a simulation package Network Simulator. Four different connection mechanisms have been realized. They are serial, parallel, pipelined and segment-filled connections. Serial and parallel connections are simulated for comparison purposes. These are connection mechanisms of HTTP/1.0. The modification proposed in HTTP/1.1 is pipelined connection. We have obtained segment-filled connection by modifying pipelined case. We have examined the performance of each modification and compared their simulation results with HTTP/1.0 connections. For the traffic conditions used in the simulations, segment-filled and pipelined connections performed better in terms of effective web page retrieval rate. In addition, as a modification to the TCP, we have increased the initial window size and compared with the one segment initial window size case. Changing initial window size from 1 to 2 and 3 has increased the performance of each connection case individually.

Keywords: Internet, HTTP, TCP, pipelining, TCP initial window size.

ÖZET

Deniz Gürkan
Elektrik ve Elektronik Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Erdal Arıkan
Eylül 1998

Bu tezde hiper metin aktarım protokolünün (HTTP) TCP ile etkileşimi göz önünde bulundurularak performans analizi için bir simülasyon çalışması yapılmıştır. Ağlararası bağlantılar üzerindeki gecikmenin azaltılması uygulama ve aktarım katmanlarındaki protokollerde değişiklikler yapılarak sağlanabilir. Simülasyonlar için, HTTP/1.0 ve HTTP/1.1 modelleri Ağ Simülatörü (Network Simulator) adlı simülasyon paketi kullanılarak gerçekleştirilmiştir. Dört değişik bağlantı mekanizması gerçekleştirilmiştir. Bunlar seri, paralel, boruhattı ve bölüt doldurmalı bağlantılardır. Seri ve paralel bağlantıların performansları karşılaştırma amaçlı olarak simüle edilmiştir. Bunlar, HTTP/1.0'ın bağlantı mekanizmalarıdır. HTTP/1.1'de önerilen değişiklik boruhattı bağlantısıdır. Bölüt doldurmalı bağlantı, boruhattı bağlantı olayını değiştirerek gerçeklendirildi. Her değişikliğin performansı denendi ve HTTP/1.0 bağlantı olaylarıyla karşılaştırıldı. Simülasyonlarda kullanılan trafik durumları için, etkin ağ sayfası erişme hızlarına göre boruhattı ve bölüt doldurmalı bağlantılar çok daha iyi performans gösterdiler. Buna ek olarak, TCP'ye değişiklik şeklinde, başlangıç bölüt pencere büyüklüğü arttırıldı ve başlangıç bölüt pencere büyüklüğünün bir bölüt olduğu olaylarla karşılaştırıldı. Başlangıç bölüt pencere büyüklüğünün 1 iken 2 ve 3 yapılmasının her bağlantı mekanizmasının performansını önemli ölçüde arttırdığı gözlendi.

Anahtar Kelimeler: Internet, HTTP, TCP, boruhattı, TCP başlangıç bölüt pencere büyüklüğü.

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my supervisor Prof. Dr. Erdal Arıkan for his guidance, suggestions and valuable encouragement throughout the development of this thesis.

I would like to thank Assist. Prof. Dr. Ezhan Karaşan and Assist. Prof. Dr. Murat Alanyalı for reading and commenting on the thesis and for the honor they gave me by presiding the jury.

I am most grateful to my dear friend Tolga Ekmekçi for his great support he gave with his presence and technical knowledge during my every desperate situation. And I would like to thank Ayhan Bozkurt as well for his technical helps in building the essential Network Simulator for my thesis work.

I am also grateful to my friends, Gülbin Akgün, Gün Akkor and Fatma Çalışkan, who made life more enjoyable during this thesis work. I am indebted to my dear friend Arçın Bozkurt for his patience and understanding in reading and commenting on my thesis. My gratitude is far less than they deserve for my lovely friends Harika Yıldız, Pınar İşçioğlu and Burçin Baytekin who made me feel lively and happy especially during the busiest period of my thesis work.

I will always feel my most special gratitude for my mom and dad because of their never-ending patience in understanding me and giving me support at all of my desperate situations throughout my thesis work. My sister, Sevgi, has a special place in my success in taking it all easy and being happier every time.

Contents

List of Figures

List of Tables

To my parents ...

Chapter 1

Introduction

Information technologies continue to expand at an ever increasing rate in recent years through the widening use of internet with huge amounts of investment in that area. The market shifts to electronic media because of its distributed nature all around the world and easier accessibility by anyone who is interested.

Internet technology has to be developed further in order to fulfill the requirements of customers in the most efficient, cheapest and fastest way. This requires the protocols handling the traffic through the world's largest information highway to be as capable as possible.

The problem of interest in this thesis is the reduction of latency through internet connections. We searched for the main problem in the connection mechanism of the application layer protocol HTTP (HyperText Transfer Protocol) and the transport layer protocol TCP (Transmission Control Protocol).

According to the OSI(Open Systems Interconnection) layered model of network structure, the application layer and the transport layer protocols are completely dependent on the congestion control algorithm. In this respect, they should be designed properly in order for the hardware properties of the network to be used in the most efficient way.

The protocol architectures are summarized in Figure 1.1 [1]. Application layer protocols have more user interaction than the other layers. Hence, they should be more flexible. In addition, the upper layers of the model described in Figure 1.1 are more

dependent on the algorithm of the software, because the lower layers are nearer to the physical layer. The reason of this is the hardware dependency. Software is constrained by the properties of the hardware. Briefly, the network has a layered architecture and each layer must be designed by taking into account the performance of the lower layers. Since application layer is the top layer, it has to consider the performance of all remaining layers.

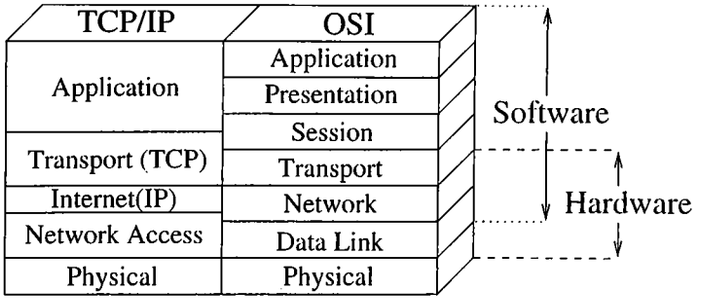

Figure 1.1: Protocol architectures.

1.1 Structure of the Protocol : HTTP

HTTP/1.0 was designed as the application layer protocol handling connections to the WWW (World Wide Web) by an appropriate browser. It operates with one TCP connection for both command and data exchange. However, HTTP/1.0 lacks the appropriate configuration in its design for today's traffic characteristics through internet. Hence, we have worked on possible improvements on the protocol.

HTTP assumes a reliable underlying transport protocol to handle the hypertext files. Most of the time this protocol is TCP (Transmission Control Protocol). Hence, it does not take any action for the network services like flow control, error control, and sequenced delivery of the data to be transmitted.

The information exchange system on the internet is based on the transmission of pages created in HTML (HyperText Markup Language). The standardized HTML is displayed by web browsers after appropriate parsing in order to display it according to the commands inside the page. The HTML has commands for the display of the page in the required manner. For example, there are commands for creating links in the page, or for displaying an image on the page. These commands are interpreted by the browsers.

They are made as simple as possible in order to create the page in the most desired manner and at the same time to have the smallest file length.

1.1.1 Message Handling

HTTP accepts any reliable protocol in the underlying transport layer. Most widespread usage is with TCP as the protocol under HTTP for internet connections. A **connection** is a virtual circuit between two application programs established by the transport layer for the purpose of communication. The application program sends a request in order to establish a connection and this application program is called the **client**. The client initiating the request can be a browser. The application program that accepts connection requests and sends back responses is the **server** [2].

Throughout the thesis an HTTP request message will be referred to simply as a **request** and an HTTP response message as a **response**.

HTTP assumes that the client side opens the connection. The client sends the request to the server and the server sends its response. Current implementation, HTTP/1.0, requires the server to close the connection after sending the request to the client [3].

HTTP uses simple commands for data retrieval. These commands are OPTIONS, GET, HEAD, POST, PUT, DELETE and TRACE. Such a small number of commands is enough for HTTP. The most frequently used methods [1] are GET and HEAD. These commands are marked as safe methods. GET makes the protocol send and retrieve whatever information is available. Similarly, HEAD takes only the message header, message body is not included. Only the GET method is used in the simulations since it is the most general method and it summarizes the traffic through the internet.

1.2 Problem Statement

The traffic characteristics of WWW is asymmetrical in terms of the amount of data sent and received [4]. It requires the client side to receive large amount of data and the server side to receive requests which are much shorter, on the order of hundreds of

[1] We used method and command for HTTP commands interchangeably.

bytes. This makes the traffic asymmetric in nature and HTTP/1.0 was not designed to handle this kind of traffic. The heavy load of the traffic comes from the more mediatic page style of web. Web sites include every kind of media files to be both attractive and informative. Examples of the media files are images, animations, audios and video files. These are included into a typical HTML file with an address. The address points to a local directory where the actual HTML page resides in. The HTML page is parsed by the browser at the client side and discovered that it has media files to be taken from the server side. Then, those inline files are requested from the server side by opening one more connection for each of them.

HTTP/1.0 makes this exchange in the following way: The client opens connection first for the HTML page. After sending the HTML file, the server closes the TCP connection immediately. The inline files parsed by the client browser are requested from the server side one by one and with individual TCP connections for each of them. Hence, there occurs unnecessary open connection and close connection phases for a retrieval. However, the process could take place within only one connection. Opening one connection is proposed in the HTTP/1.1 [2]. It is called as pipelined connection in [2]. The connection opened for the HTML file is used for retrieval of the media files inside that page.

1.3 Outline of Work Done

We have investigated the pipelined transmission performance over HTTP/1.0 over only one connection. Pipelined transmission, as summarized above, is proposed as an improvement from the serial retrieval of data [5]. Further, we have run simulations on segment filled pipelined transmission as a new contribution to the research area.

The procedure outlined as retrieval through only one connection is called as pipelined transmission. Pipelined transmission should take into account of the congestion control algorithm of TCP, which is outlined in Chapter 2. When congestion control algorithm of TCP is considered, the pipelining method can be modified even further. We have proposed to use segment filled pipelining. This modification and true pipelining are investigated through models built under NS (Network Simulator of Lawrence Berkeley Lab). The models are described in Chapter 3.

In addition, we have run simulations with the models on the performance of the initial window size increase of TCP. That modification is on the transport layer of the OSI model. The results illustrated considerable amount of improvement as outlined in Chapter 4.

1.4 Organization of The Thesis

Chapter 2 contains background information on the Internet protocol architecture, the congestion control algorithm of TCP, and structure of HTTP. In Chapter 3, there is a summary of current observations and research made in order to improve performance of HTTP/1.0. In addition, we have included a summary of our simulation study and models used in the experiments in the same chapter. Chapter 4 has simulation results and discussions on the results. The last chapter has conclusions about the work and directs to some future study about the subject.

Chapter 2

Review of Internet Protocol Architecture

In this chapter, the Internet protocol architecture will be presented. The Internet protocol architecture is composed of TCP/IP on the transport layer and network layer. It has HTTP on the application layer. Current version of HTTP is HTTP/1.0 [6]. There is a prospective standard proposed in [2], the HTTP/1.1. Since we aim to improve the performance of HTTP, its operation plays a key role in our discussions. Therefore, operation of HTTP/1.0 will be reviewed. In addition, a summary of the congestion control algorithm of TCP is included since it affects the performance of HTTP directly. The traffic conditions that are present in the design of modifications to HTTP are also very important in building the models. Long observations of data flow through internet give reliable empirical models for the traffic conditions of internet. We have included those models in this chapter.

2.1 Operation of HTTP/1.0

HTTP was designed by taking the available protocol FTP (File Transfer Protocol) as a basis at the very beginning. FTP works with two TCP connections at a time; one for

command exchange, and one for data transfer. This scheme is appropriate for FTP traffic because FTP file transfer is mostly for large files and command exchange is with small length data. The file transfer with FTP causes TCP's congestion control algorithm work in its full efficiency. There are usually large files to be transmitted by FTP. However, this is not the case with commands. In the command exchange, only the slow start phase of TCP will be activated for relatively shorter data lengths. This will manage the transmission in the required speed.

HTTP imitated FTP by its simplicity of methods. It is designed to be simpler than FTP with its number of connections. HTTP handles only one TCP connection for each client-server pair. The data from client to server is like the command data of FTP since it is composed only of requests. On the other hand, the traffic from server to client is much heavier in that it is composed of the files requested by the client. Hence, internet traffic is not symmetrical. The server to client traffic is always much heavier.

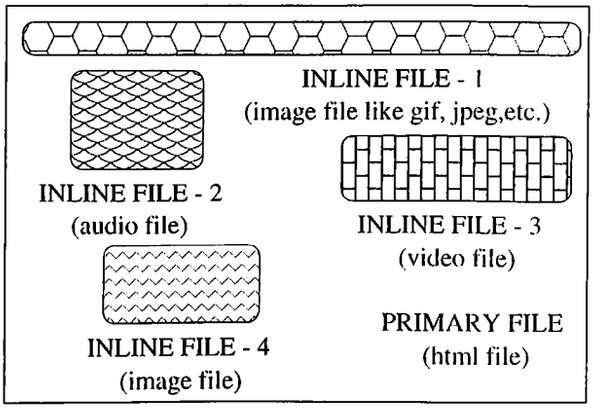

Figure 2.1: Typical page retrieved through internet.

At the very beginning of the design of HTTP, the traffic was not composed of a variety of different objects other than HTML files. HTML files are pure text, hence only ASCII data transmission is required. Today, internet traffic handles image, audio and video files in addition to the text of HTML files. Figure 2.1 shows the typical page retrieved by a browser. It includes what we call inline files in image, audio or video format in addition to the main file, called primary. Internet traffic was nearer to FTP traffic before these multimedia objects took place in the WWW. As the available bandwidth increases with new hardware and transmission technologies and as the Internet is used more for every kind of information exchange, the number of inline files increases. Hence, it becomes unsuitable for HTTP to handle this new type of traffic, which it was not designed for.

The page structure illustrated in Figure 2.1 is retrieved through HTTP/1.0 by opening one connection for each inline files inside the single internet document. From this point on, we will refer to the whole page with its primary and inline files as **document** (or **web document**), and any file like primary or inline simply as **file** in the web document. The sequence of states of the connection are illustrated in Figure 2.2 during the retrieval of a document.

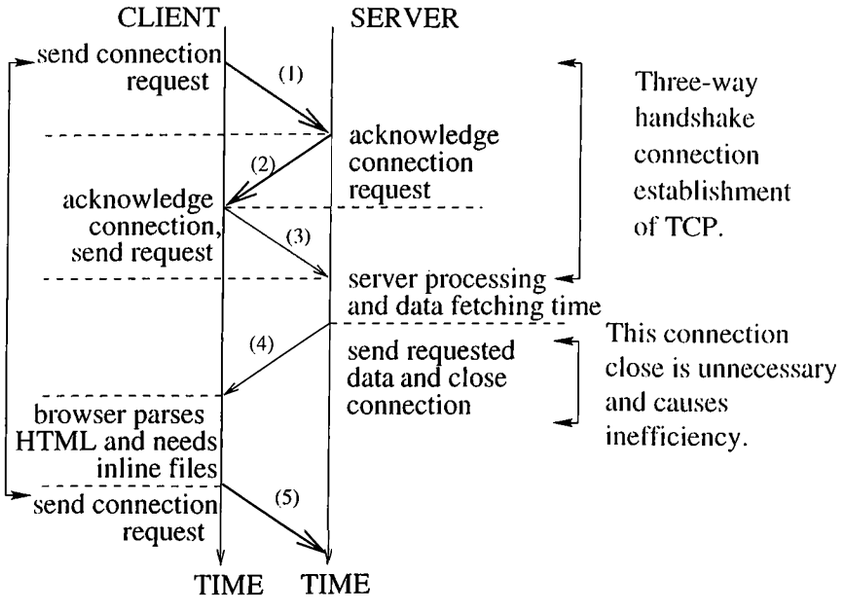

Figure 2.2: Typical restart of TCP connection for retrieval of one page.

Figure 2.2 summarizes the overhead and delay introduced by opening individual connections for each of the inline file in addition to the primary for single web document. The three-way handshake connection opening is iterated for each inline after retrieval of the primary file. The structure of HTTP/1.0 requires the server closing the connection after requested data is sent to the client side. The close connection is unnecessary in that case if any inlines exist in the page. Hence, if the client needs inlines, the open connection process restarts. There will be 3 round trip time delays added to the retrieval of the document. In addition, the restart of TCP prevents it from reaching its most efficient phase of transmission.

If we could make the server stay idle for some time after sending the requested data, it could answer other requests for inlines without the need for an additional open connection. Moreover, the congestion control algorithm of TCP could have been used more

efficiently in that case. There will be larger amounts of data transmitted through one connection end to end. This procedure is proposed as pipelined connection in HTTP/1.1.

2.2 Traffic Conditions Handled by TCP

TCP (Transmission Control Protocol) is the connection oriented protocol for the transport layer of the OSI layered structure. It is designed to provide reliable communication between pairs of processes (TCP users) across a variety of reliable and unreliable networks and internets [1]. The reliability of TCP connections supply the transport layer the flow control, error control and sequenced delivery. These controls and sequenced delivery is made possible by the help of the connection-oriented nature of the protocol [7].

We will refer to the unit of transmission with TCP as **segment** or as **packet**. Accordingly, the quantity referred to as **window length** is the number of segments.

TCP guarantees to give reliable service to the end users by its congestion control algorithm. The algorithm is designed for this purpose. In order to have a reliable communication between the end users connection establishment and termination states are included in the state transition structure of the protocol. The connection establishment state guarantees that a virtual circuit is present between the end users so that each of them listens to one another. In addition, the end users can exchange their parameters of transmission through the connection. There exists a well defined retransmission strategy. The packets are delivered in order. Although it is complex, connection timeouts or hardware problems can be announced to each end user for most of the time.

The unreliable counterpart of TCP is UDP (User Datagram Protocol). This protocol differs in its handling of retransmissions and order of data. Since this kind of transport protocol is not connection oriented, no connection establishments or terminations exist. The data delivery in sequenced form and duplicate protection are not guaranteed, hence there remains so much to do for the upper layer protocol in that case. The only advantage of it is its low overhead. Since HTTP assumes the underlying transport protocol to be reliable, the lower overhead unreliable UDP is not appropriate for HTTP. Hence, we used TCP as the underlying transport protocol in our simulations.

2.2.1 Congestion Control Algorithm of TCP

TCP uses a congestion control algorithm of [8] in controlling the length of streams of data transmitted between end users. This algorithm works according to ack (acknowledge) traffic coming back from each end user. It uses windows as the unit of transmission at each time. Window size is determined according to ack flow traffic of the connection [9]. It is increased if ack's for the stream of data in the window is sent, and it is decreased to its initial value if a packet drop is sensed [10].

The algorithm has two phases:

1 - Slow Start Phase

2 - Congestion Avoidance Phase

The algorithmic summary of the phases is outlined below [8]. Taking current window size as W and a threshold as W_t, the updating mechanism can be summarized with the following statement.

if $W < W_t$, set $W = W + 1$,
else set $W = W + 1/W$.

If a packet loss is detected:
set $W_t = W/2$,
set $W = 1$.

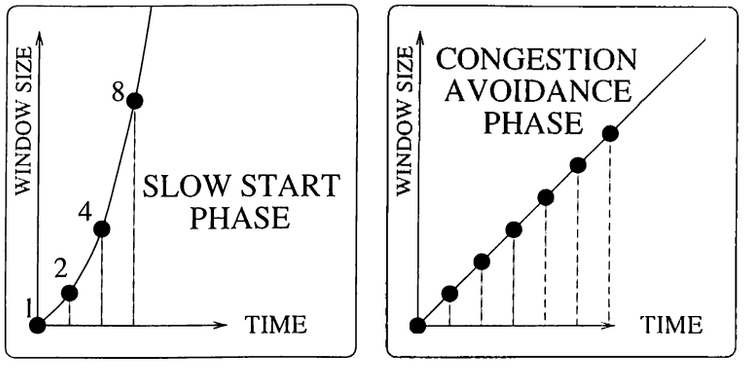

Figure 2.3: Congestion Control Algorithm of TCP - Phases.

1 - Slow Start Phase : The initial window size is generally taken to be one unit of maximum packet size defined for that connection. In the slow start phase the window size increments for every successfully acknowledged packet until it reaches half of the

window size at which the last packet loss was encountered [8]. This phase makes the window size increase in a somewhat exponential manner, contrary to its name. This phase continues until $W = W_t/2$ is reached or a packet loss occured.

2 - Congestion Avoidance Phase : After window size reaches a number larger than the threshold W_t value, this phase begins. The demand for extra bandwidth with this algorithm continues by incrementing the window size by one for every window's worth of acknowledged packets [8]. This increase is more moderate than the one in the slow start phase. This phase continues until a packet drop is seen. The window size at the packet drop event is used for determination of the next threshold value.

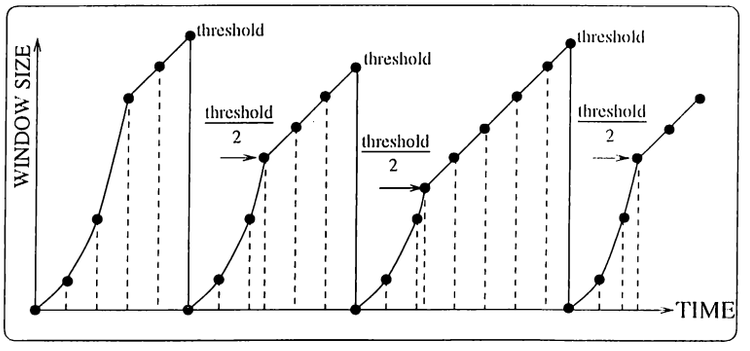

Figure 2.4: Congestion Control Algorithm of TCP - Typical Run of Algorithm.

2.2.2 Traffic Characteristic Best Fitting TCP

In order to utilize TCP congestion algorithm in the most efficient way, the transmission should have a duration long enough for the TCP congestion control algorithm to reach its congestion avoidance phase. The algorithm reaches its congestion avoidance phase after some number of end-to-end delays depending on the network conditions. If there is congestion in the network, the window size will be shrunk to the initial size more often. This will result in a number of unwanted slow-start restarts [11]. The traffic handled by TCP should be a persistent connection with large bulk data transfer for most efficient transmission. This will give the chance to the congestion control algorithm to enter the congestion avoidance phase in a relatively short time. The window size will increase consistently in that phase. This is illustrated on the left plot of transmission of Figure 2.5. The window size will increase consistently in that phase. Since the window

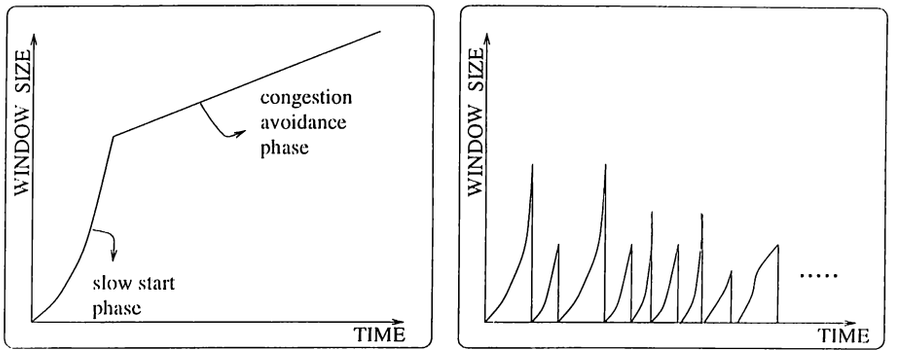

Figure 2.5: Ideal (left plot) and inefficient (right plot) cases for a TCP connection.

size is increased to a large value, the bandwidth demand is also persistent [12]. Hence, the transport protocol demands for extra bandwidth by taking feedback from the network.

If the traffic at the time of transmission is short bursts of data, then congestion control algorithm will get stuck in slow-start phase and the threshold will be lower. It will be inefficient to send the data with windows in that case. This is illustrated on the right plot of Figure 2.5. Congestion control algorithm works in small length windows and the link is not used in its full utilization. Since there is no demand for extra bandwidth, small packets are sent end-to-end with small window length.

2.3 Requirements of Application Level Protocols

Internet traffic requires the underlying application level protocols to handle small bursts of traffic with relatively more frequent requests than FTP. The requests occur in small intervals. Internet browsers (like Netscape, Microsoft Internet Explorer, etc.) parse the HTML page received from the server and determine what to request next if any media file (like image, audio, animation files) exists in the page. This usually lasts for a few milliseconds in the parsing machine of client side and less than a second with the end-to-end delay of the network until it reaches the server side.

As the above characteristics of traffic imply, the application level protocol should be working fine with these waiting intervals. However, there is the other side of the story. The protocol should also handle interactions with the underlying transport layer protocol by taking into account of the transmission performance effects. In the internet

transmission the transport layer protocol is TCP most of the time. As mentioned above, TCP is not designed for internet data traffic.

Web documents include inline files that should be transmitted when required from the client side. HTTP/1.0 makes this transmission in a serial manner [13]. This introduces an unwanted delay for the user of the system. Actually, the reason of the delay is unutilized usage of the link in between. Internet traffic is similar to the one on the right plot of Figure 2.5. The required bandwidth for the transmission of internet files is kept small with serial transmission scheme. Hence, the utilization of the link will be low.

The application level protocols deal with users interactively contrary to the case for the lower layer protocols. The web traffic consists of retrieval of web documents. It requires different file formats to be handled by the protocol. It works in a transaction-oriented manner. Every request reply pair is treated independently. The connections are also handled as a one-to-one dependent events to the transaction. This is not a strict requirement. A connection may handle more than one transaction in order to service more than one file request-reply pair.

2.3.1 Asymmetric Nature of Internet Traffic

WWW is known as the world's largest distributed file system. Users connect to different servers all around the world and retrieve different length documents. This process cannot be modeled by well-known distribution functions [14]. Hence, the requirements of bandwidth reduction and server load in addition to quicker retrieval of data should be satisfied by extensive investigation of traffic conditions through the Internet.

The client side is busy with sending of requests to the server side during a typical transaction. The servers have heavier load in a typical connection. Therefore, internet traffic is specified as asymmetric with more load on the server to client traffic. The servers should handle a large number of connections. At the same time they should run a database management program fetching the requests of clients.

Caching is done either on the client side or on the server side. Caching at both sides is also possible. Caching means taking local copies of web documents at different locations. Caching at the client side is done inside the user hard disk, or to a LAN (Local Area Network) disc. Caching at the client side gives the opportunity of fast retrieval of previously retrieved files. The client side copies all web documents into the local

directory of the user. The file is retrieved from that local copy if it is not modified by the server side until that next retrieval.

At the server side caching is done inside the proxies. Proxy is any machine behaving like the server in order to ease the load of the server. Hence, copies of most frequently used web documents are stored in those proxy machines. Requests for those documents can be forwarded to those machines for faster retrieval.

Chapter 3

Literature Survey and Simulation Models

After examination on the latency in HTTP/1.0 [11], research has been done for latency reduction by [5] and [15]. Modifications were proposed with this respect. In order to propose the most covering modification for the latency reduction problem of HTTP, the Internet traffic conditions should be investigated extensively. For this purpose, we have included model of the traffic handled by HTTP in this chapter. Then, we have outlined the historical sequence of propositions for reduction of latency.

3.1 Model of the Traffic Handled by HTTP

In order to design the appropriate protocol for internet application layer, traffic characteristics should be observed. If we can model the traffic through internet connections, the protocol requirements can be determined in a more realistic way. Similarly, the simulations will be more reliable in that case.

A model is investigated in [4] fitting to the following parameters which are used in our simulations:

Request length :	HTTP request length.
Reply length :	HTTP reply length (primary and inline).
Document size :	Number of files per document.
Think time :	Time between retrieval of two successive documents.

In our simulations the request length is kept constant as 210 bytes per file. This size is taken from [4] following its result that medians of the request length is 240 bytes.

The reply lengths are determined according to the file to be transmitted. It can be the primary file of a web document or one of the inlines in that document. The sum of them gives the total length of the web document to be retrieved. According to [4], the reply lengths can be modeled as a Pareto Distribution[1] with parameter α.

The model is the approximation to empirical data taken from traffic monitoring programs. This data is collected by observing the retrieved file lengths during connections to some relatively busy servers. This data is then processed and the characteristics are investigated.

The most important and most attractive nature of WWW is its lack of rules in designation of a page. A page can be constructed for any purpose and it does not follow any rule like "There should be a header on top, there should be chapters, etc..". The fast and unpredicted growth of WWW is in some ways seen as a result of its style-free nature. In fact, if there were strict rules that should be applied to a page design, WWW would not be so convenient to use for so many different people and different purposes. Hence, the file lengths vary in the most unpredictable way. It is not surprising to have a model that fits to Pareto distribution in that case. The Pareto distribution exhibits near infinite variance which fits the discussion we have made above.

The α parameter in the Pareto distribution function was set to be in the interval $[0.85, 0.97]$ for the primary replies and $[1.12, 1.39]$ for the inline replies. We took the medium value so that $\alpha = 0.91$ for primary file lengths and $\alpha = 1.24$ for the inline file lengths. The other parameter which is known as the minimum value of the distribution is taken as 1 kbytes as found out empirically in [4].

From the observations done in [4], it was found out that the number of files per

[1] The heavy tailed Pareto Distribution has a cumulative distribution function given by $F(x) = P[X \leq x] = 1 - (\frac{k}{x})^{\alpha}$, where k is the minimum value of X and α is the shape parameter.

document is less than four files for 80% of the documents. Hence, we took 3 inline files and 1 primary file as a web document throughout our simulations.

3.2 Connection Improvements by HTTP/1.1

To prevent the latency caused by serial connection of HTTP/1.0, the pipelining method was proposed by [16]. The latency problem of HTTP is outlined and transmission problems causing the latency is investigated in [17]. The pipelining method of connection is experimented in [15] and [18]. The pipelining method is proved to yield well throughput with those experiments. The research on pipelining method is continued by [5]. An extensive study was presented in that report. The results demonstrated that the performance of HTTP connections can be improved considerably by pipelining.

Meanwhile, the effects of TCP slow-start for files of a web document were investigated in [13]. It included a demonstration of the timetable of events during a typical web page retrieval. That paper addressed the solution of the problem to pipelining method in connection.

3.2.1 TCP Modifications for HTTP

The TCP-HTTP interaction is investigated with experiments further in [17]. As a result, transaction TCP (T/TCP) and pipelined connection HTTP performed better. T/TCP is outlined in two RFC's, one referring to its concepts [19] and the other to functional specifications [20]. Some modifications are proposed with T/TCP in order to handle transaction-oriented nature of HTTP with [19] and [20].

Distributed applications in the internet use a transaction-oriented style of communication rather then virtual circuit style. The applications on the internet have a request-response type of communication. This traffic is not appropriate for the design and usage of the congestion control algorithm of TCP, since TCP has to be used with large data transfer requirements [21]. Hence, the transaction-oriented internet application should suffer the overhead of opening and closing TCP connections. If the application chooses not to have that overhead, there should be an application-specific transport mechanism built on top of the connectionless transport mechanism UDP [19].

The transaction-oriented service is an interaction with a request followed by a response. Hence, an explicit open or close phase would impose excessive overhead. The simplest transaction would be one with single-segment request and single-segment response. But TCP implementations use at least 10 segments for this sequence in a transaction. These are: 3 segments for 3-way handshake opening the connection, 4 segments to send and acknowledge the request and response data, 3 segments for TCP's full-duplex close sequence [19].

The TCP transfers have a symmetrical nature. They are composed of data transfer and acknowledge of that data. The symmetry is seen in the sequence numbers of duplex traffic. On one way goes the data, and on the other way goes the acknowledge segment with the same sequence number as the data. For request response sizes small, the transaction becomes dominated by the connection open and close phases. But when the request and response messages increase in size, the symmetry again dominates and the open and close phases do not contribute much to the total communication length in percentage overhead [19].

The transaction TCP is a new version for TCP with modifications proposed as a solution in order to handle transactions in a more efficient way. It is preferred to have a new version of TCP to handle transactions rather than having a new protocol specifically adapted to transactions. One modification proposed is bypassing the 3-way handshake. Bypassing the 3-way handshake is done by long lasting transactions, with idle connections. That is, the connection is kept open even if no data transfer is taking place. Hence, the next data transfer is done on the already open connection. The other modification is shortening the TIME-WAIT state delay of TCP. This state is entered after the connections is closed. The connection information is kept stored for 4 minutes at the end users. This is a significant overhead for busy servers of internet. Hence, they keep this delay time as short as possible [19].

Other than improving the restart of ideal TCP connections in pipelining method, TCP initial window increase was suggested in [22]. This intends to send more data at the startup of a connection. This study was published as an internet draft [22]. It was investigated with a rough model of HTTP/1.1 and parallel connection HTTP/1.0. We have added real pipelined and serial connection performance to that study. The results are in Chapter 4. Initial window size of a TCP connection is denoted as IW. The increased IW proved to improve the performance for all cases with $IW = 2, 3$. As the window size increases further, congestion effects overcome the persistent transfer, and

the performance degrades.

Pipelining method is implemented for simulations done in [22] and the restart of idle TCP connections is examined by [23]. There were two approaches for the restart of idle connections in [23]. The congestion control algorithm may be forced to restart with a window size of one, or it can use the last window size just before the connection becomes idle. The first approach is very conservative. It saves the round trip times spent for a 3-way handshake open connection phase. On the other hand, it restarts with window size of one without checking for extra bandwidth. The second approach is on the contrary too much demanding. It assumes that the link stayed that much congested throughout the idle period such that the last window size will be appropriate again. But it may not be the case. The paper [23] proposes a method called rate-based pacing (RBP). The idle TCP connection restarts with a moderate window size and continues with feedback of round trip time form the network. It restores to its last window size with the help of the frequency of acknowledge packets received in turn.

3.2.2 Pipelining Method in HTTP

The pipelining method is then included in the new version of HTTP/1.1 prepared as an RFC document in [2]. The previous version, HTTP/1.0 was documented in [6]. After the release of the RFC document for HTTP/1.1 cited in [2], the research and experiments continued on the modifications suggested for HTTP/1.0.

There were experiments done on pipelining method in the technical note [15]. All experiments in that note pointed that the pipelining method is more efficient than the serial connection HTTP/1.0.

The pipelining method was investigated over different transport protocols in [18]. The authors in [18] used parameters like server processing time and latency definitions in order to analytically investigate the performance of HTTP over different transport protocols.

In addition to pipelining, we investigated a method with equal length segments, what we call segment filled connection. The results showed improvement when compared to pipelining. The segment filled method of transmission requires some information to be transmitted to the client side regarding the length of files transmitted end-to-end in a glued fashion.

3.3 Simulation Models

We have done simulations in order to investigate the performance of the methods outlined above with models of HTTP/1.0 and HTTP/1.1 the results of which are summarized in Chapter 4. The models used in [22] are improved to cover the exact behavior of pipelined HTTP. We have built the models for the serial connection, true pipelined connection, segment-filled pipelined connection and parallel connection HTTP.

We ran our simulations with the Network Simulator of Lawrence Berkeley National Laboratory. The simulator can be downloaded and installed to appropriate operating systems from the address mash.cs.berkeley.edu/ns/ns.html.

We have used the models under www-mash.cs.berkeley.edu/ns/ns-contributed.html, the work titled "Simulation Studies of Increased Initial TCP Window Size" as basis. On top of them we have developed our own modifications.

The simulation topology and the performance criterion are presented in the next chapter. Also included are the simulation results with the models built for different connection cases.

Chapter 4

Performance Evaluations

We have done simulations on the modifications suggested in Chapter 3 in order to reduce latency in HTTP/1.0. The simulations can be listed under four main titles.

1.**Serial connection HTTP/1.0** is simulated for comparison reasons since it is the current standard.

2.**Parallel connection HTTP/1.0** is the temporary solution found for latency reduction of HTTP/1.0. It is appropriate for some powerful servers and high bandwidth links.

3. Simulations are done on **pipelined HTTP/1.1** under various traffic conditions.

4. We proposed **segment-filled pipelined HTTP/1.1** as a further modification.

In addition, we have run simulations on TCP initial window size increase for the cases outlined above and evaluated its effect on performance.

4.1 Network Topology and Traffic Characteristics

In the simulations we used the topology in Figure 4.1 for the test network.

This topology summarizes a typical client server connection for internet purposes.

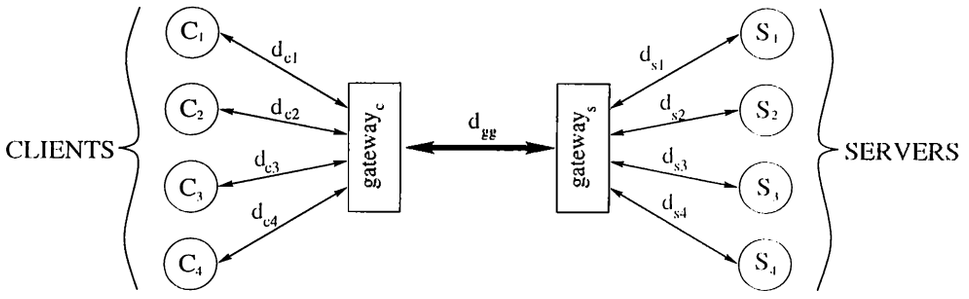

Figure 4.1: The topology of the test network used in simulations.

Client side represents a LAN with an outgoing gateway and the server side represents another LAN with its gateway. Typically, the gateway to gateway links are the most congested ones with longer delays than the links inside a LAN.

Throughout the simulations we connected 2, 4, 8, or 16 web client agents to each node at the CLIENTS side. This resulted in simulations with 8, 16, 32, and 64 web clients, respectively. In order to obtain the worst case performance analysis, we added a background traffic with 1, 2, or 3 FTP client agents. The FTP traffic consisted of transmission of 1-Mbyte of data persistently, that is, during the simulation time, there always existed 1, 2, or 3 1-Mbyte file transfers through the same link. This background traffic helped us obtain various utilization percentages of the link. Hence, we tried to demonstrate the performance of modifications on different congestion situations.

We used duplex links and assigned 1.5 Mbps as link capacity for the gateway to gateway link. The link capacity of client to gateway and server to gateway links were 10 Mbps as it would be in the case for an Ethernet connected LAN. Link delays were assigned as follows:

d_{gg}=50.0 msec, d_{c1}=1.0 msec, d_{s1}=100.0 μsec,
d_{c2}=2.0 msec, d_{s2}=2.0 msec,
d_{c3}=3.0 msec, d_{s3}=3.0 msec,
d_{c4}=4.0 msec, d_{s4}=4.0 msec.

The gateways have buffer capacity of 25 packets. TCP connections have segment (or packet) size 1460 bytes. The initial window size of TCP connections is 1 packet. In addition, we have run simulations in order to see the performance of the protocols

when the initial window size is increased to 2 or 3 packets. Those results are included in Section 4.7. In any case, throughout all the simulations a dropped packet causes the window size to shrink back to 1 packet as in current practice of TCP connections.

For performance measurements we used the effective web page retrieval rate and packet drop rate. We defined effective retrieval rate as the ratio of total bytes retrieved through web connections or FTP connections to the time of retrieval of those pages. The simulation of typical connections of web consisted of retrieval of primary file and retrieval of inline files. The consecutive web document retrievals start after a nonzero interval called **think-time** for the web client to initiate a new connection. In the simulations we assigned think-time from a uniform distribution for the interval 1.0 msec to 2.0 msec. The simulation of a file transfer with FTP composed of a number of FTP clients ranging from 0 to 4 each of which persistently requested 1 Mbytes of data from the server immediately after one file transfer finished. This made the link highly utilized and supplied the adequate simulation environment for the performance analysis. We ran the simulations for a long duration, about 300.0 seconds, to ensure an appropriate sample for the long run behavior of the link. The simulation results are tabulated for those simulations in the corresponding sections.

4.2 HTTP Latency Reduction

Current standard for HTTP is HTTP/1.0. Before the new standard HTTP/1.1 was thought of, some small modifications were made on HTTP/1.0. Rather than opening individual connections to each file transfer of a single document, the improved HTTP/1.0 can implement parallel connections. That is, after determining the inline files to be requested from the server, client side opens one connection for each inline and sends the requests through those individual connections. This connection opening sequence has short time intervals like 1-10 msec. During transfer of each inline, connections for them stay alive and hence the connection is called parallel. The serial connection case opens a new connection when previous file transfer is finished.

Although this parallel connection establishment is a very promising method in reduction of latency through internet connections, it has a high demand on server and client side ports. E.g., there will be an overhead of 3 connection establishments for an internet document of one primary and 3 inlines. Since the connections will be alive for

the duration of each of the inline retrievals, the network usage will be three times as much for 3 inlines when compared with one connection for all of them. This method is not appropriate for very busy servers. The number of connections alive at a time is an important parameter. It is more important than how long each of them lasts. This implies that parallel connection opening will be a heavy load on the servers. Since each connection corresponds to a transaction from the server side point of view, each transaction will have a record in the server machine. The server has the responsibility of taking the necessary information about each transaction. Hence, opening more than one connection for one page retrieval will be an inefficient process for the server.

4.3 Serial Connection HTTP/1.0

HTTP/1.0 opens one connection for each file requested. Each connection lasts for the duration of retrieval of each file by the client side, and ends by the one-sided decision of the server to close the connection. Hence, the client needs to open another connection to the server if it discovers any inline file for a document from that same server.

Figure 4.2 shows such a connection scenario with the file request times and lengths indicated on the right of Figure 4.1.

The simulation in Figure 4.2 is done with one web and FTP client in the network. The simulation is run for 30.0 sec serial connection of HTTP/1.0 with 3 inline files. The diamond markers on the plot in Figure 4.2 give the window length at transmission time. Since we have set the maximum segment size to 1460 bytes the markers are a multiple of 1460 except for the last segments of files.

Figure 4.2 shows how individual connections are opened for each request. The server side closes the connection after sending the requested data. The client opens another connection for the same document's inline file although it will again be retrieved from the same server. In Table 4.1 we can trace the file lengths of primary and inline files and their request times at the server side. The web documents retrieved have length primary + 3 inlines. But the serial retrieval of them requires 4 connections to be opened rather then one connection for the whole document.

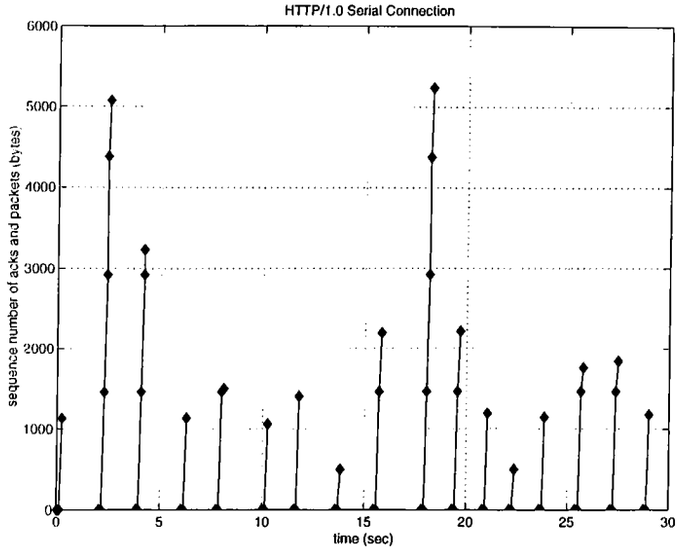

Figure 4.2: Serial connections in current HTTP/1.0. Each connection is represented by a line. Diamond markers indicate the sequence number of the TCP segments.

time(sec)	length(bytes)
0.17	1128 primary
2.21	5073 inline
4.00	3231 inline
6.20	1130 inline
7.91	1498 primary
10.17	1056 inline
11.71	1402 inline
13.75	500 inline
15.62	2191 primary
17.97	5236 inline
19.49	2210 inline
20.98	1187 inline
22.31	500 primary
23.78	1136 inline
25.57	1757 inline
27.29	1841 inline

Table 4.1: Primary and inline file lengths and their request times from the server.

4.3.1 Simulation Results and Discussions

Table 4.2 summarizes the simulation results for the HTTP/1.0 serial-connection case under the various traffic conditions outlined in Section 4.1. The simulation results imply that effective web page retrieval rate for serial connections stay upper-bounded by approximately 10 kbytes/sec for the link under consideration. It is interesting that the link utilization stays at approximately 10% for the case where there are 8 web clients and no FTP client. Since the connection open and close communication packets are 40 bytes header only, the link is not used in its most effective way for most of the transmission time. This is also seen in each of the no-FTP-client cases where web page retrieval rate is around 10 kbytes/sec. Since the file sizes through internet traffic are shorter, TCP connections do not stay alive so long that they can require extra bandwidth from the link. The transactions are composed of a couple of requests and responses in that case. TCP has a connection close and open phase that has a couple of requests and responses. When compared with the traffic during those phases with the transaction itself, the phase messages dominate the traffic. Hence the transaction lasts for so short that the

Table 4.2: Serial Connection Case - Simulation Results

# web clients	# ftp clients	percentage link utilization	effective web page retrieval(bytes/sec)	percentage packet drop rate
8	0	8.8%	10368.03	NONE
8	1	89%	9159.99	NONE
8	2	98%	5100.09	0.13%
8	3	98%	4087.44	1.02%
8	4	95%	3146.17	1.78%
16	0	18%	10304.96	NONE
16	1	92%	8452.32	NONE
16	2	96%	4986.42	0.60%
16	3	98%	4027.13	1.42%
16	4	98%	3483.44	2.26%
32	0	34%	10016.02	NONE
32	1	94%	7051.86	0.05%
32	2	96%	5129.04	1.29%
32	3	99%	4156.02	2.39%
32	4	99%	3598.56	3.11%
64	0	69%	9305.57	0.07%
64	1	93%	5582.21	1.87%
64	2	97%	4313.85	3.30%
64	3	99%	3420.34	4.72%
64	4	99%	2968.04	6.38%

overhead dominates the request-response interaction.

Packet drop does not occur for the cases of 8 and 16 web clients with 0 and 1 FTP client and for the case of 32 web clients with no FTP client. Since serial connection opening has a more conservative way of using the link, drop rate does not increase so much as the link is utilized further. With the 64-web-clients case, link utilization reaches its maximum and, not surprisingly, packet drops occur. For large number of web clients the effective retrieval rate reduces to less than 3 kbytes/sec. The FTP traffic gains control of the link capacity in that case with its large length data transfer. The TCP congestion control algorithm demands for extra bandwidth through FTP traffic. Window length expands and there will remain much less bandwidth for internet traffic in that case.

4.4 Parallel Connection HTTP/1.0

Parallel connection opening occurs during inline transmission of a document. After fetching the primary file, the browser of the client decides on the inline addresses of the document during the parsing and display of the document to the user. The browser then opens individual connections to the server side for each inline to be retrieved. These connection-opens have durations on the order of milliseconds depending on the capability of the client's processor. At the end, the connections are almost in parallel when their opening time differences are compared with the transmission time of each inline file. The simulation in Figure 4.3 is done with one web and FTP client in the network. The

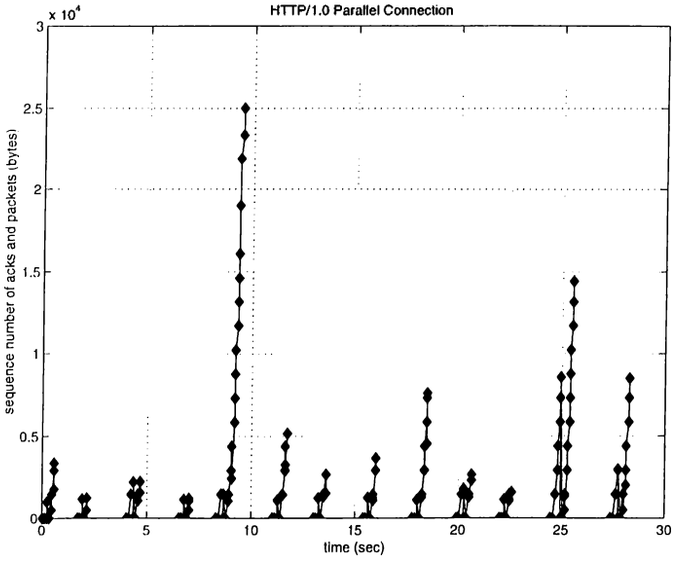

Figure 4.3: Parallel connection in current HTTP/1.0. After primary, inline retrievals are plotted nearly on top each other because of the small time difference between requests for them.

time(sec)	length(bytes)
0.177	1004 primary
0.391	3370 inline
0.392	500 inline
0.393	1789 inline
1.858	1187 primary
2.074	500 inline
2.075	500 inline
2.076	1252 inline
4.184	2217 primary
4.509	2235 inline
4.511	1565 inline
4.512	1077 inline
6.704	1148 primary
6.941	1216 inline
6.942	500 inline
6.944	1074 inline

Table 4.3: Parallel connection sample of primary and inline file lengths.

simulation is run for 30.0 sec parallel connection of HTTP/1.0 with 3 inline files. The diamond markers on the plot in Figure 4.3 give the window length at transmission time. Since we have set the maximum segment size to 1460 bytes the markers are a multiple of 1460 except for some last bytes of files.

It is noticed in Table 4.3 that inline requests are sent with very little time differences, such as 1 msec or 2 msec. Therefore the plot of parallel transmission of those files in

Figure 4.3 becomes inseparable from each other. The graph has collection of lines on top of each other for the inline transmissions at those instances.

4.4.1 Simulation Results and Discussions

Under the traffic conditions outlined in Section 4.1, simulations were run. The results are summarized in Table 4.4. The simulations indicate a 50% performance improvement

Table 4.4: Parallel Connection Case - Simulation Results

# web clients	# ftp clients	percentage link utilization	effective web page retrieval(bytes/sec)	percentage packet drop rate
8	0	24%	15707.56	NONE
8	1	91%	12437.93	0.05%
8	2	96%	7186.58	0.73%
8	3	98%	5726.92	1.59%
8	4	99%	4475.63	2.27%
16	0	48%	15353.96	NONE
16	1	88%	10037.12	0.74%
16	2	96%	6757.99	1.74%
16	3	98%	5293.94	2.61%
16	4	98%	4343.31	3.64%
32	0	83%	10391.18	1.16%
32	1	93%	6676.18	3.18%
32	2	97%	5159.73	4.27%
32	3	97%	3907.59	5.22%
32	4	99%	3444.44	6.40%
64	0	93%	3891.59	9.13%
64	1	98%	3332.88	10.78%
64	2	98%	2767.89	11.90%
64	3	98%	2466.48	13.21%
64	4	98%	2214.87	14.46%

in terms of effective web page retrieval when no FTP traffic was present when compared with serial case of 8-web-clients. This web page retrieval rate is increased because of the parallel transmission. Although we have 8 web clients, 24 connections are opened at the

time of inline retrieval. But, parallel transmission demands extra capacity and fails to show the same performance when background traffic is denser. The 64-web-clients case simulations showed such high drop rates that the network can be regarded unreliable. Their high drop rates result not completely from the parallel transmission style but from the congestion effects. This can be explained by the high utilization percentage of those cases.

The lowest web page retrieval rate is near 2.2 kbytes/sec which is approximately 25% less than the case for serial connection. Connection opening and closing communications put large overhead on the transmission of web pages for large number of web clients. The overhead will be present same as serial case. Parallel connection case will have same number of open and close connection phases. Hence, the overhead will be same. The advantage is that link is utilized better and the demand for extra bandwidth is supplied by large number of connections. The handicap is the capacity of servers in this case. In addition, if there exists background FTP traffic, link becomes already utilized to at least 90%. The web page retrieval rate could not reach any larger than 3.8 kbytes/sec for the 64-web-clients case. This rate is nearly 1/3 of the retrieval rate in serial connection. The reason for this performance degradation is the large drop rate. The 64-web-clients case has the potential of opening 192 connections with small transactions at a time.

As a result, with parallel connection experiments, we cannot avoid opening and closing connections. This causes the network to be used inefficiently for small length file transfers. The number of connections does not change when compared with the serial case. Furthermore, the parallel connection openings add extra bandwidth demand from the link. This makes the highly utilized link experiments end up with less performance than serial case. This is most easily seen when we compare the 64-web-clients simulation results of the parallel case with the corresponding ones of serial case.

4.5 True Pipelined Transfer in HTTP/1.1

True pipelined transfer is the method proposed in HTTP/1.1. With this method, the requested files are transmitted through the same TCP connection. After the primary file is transmitted, the server waits some more time for the client to request inline files if any present. The request is made through the same connection and the response takes place also on that same connection. Since the file length may not be a multiple of the

TCP segment size, the final segment of each file can be less than the TCP segment size. The situation is illustrated in Figure 4.4.

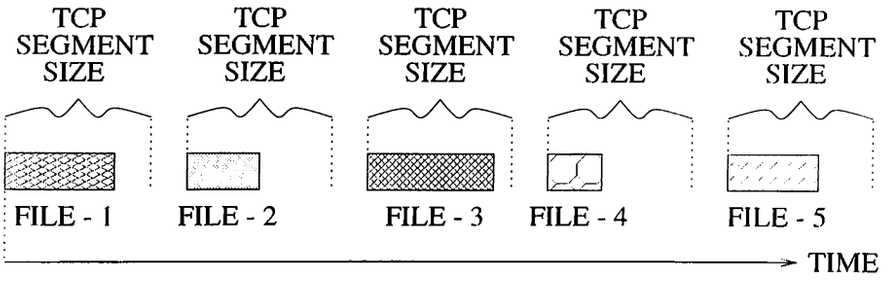

Figure 4.4: Pipelined transfer through TCP's window algorithm.

Figure 4.4 shows the last packets from the pipelined transfer of a file in a document of length primary + 3 inlines. The connection then stays open for some time set by the server's capability of handling long transfers. The client may discover inlines in the document and make its request on that already open connection. The restart of idle TCP connections were investigated in [23]. The window length is a parameter to be decided on at the restarts of TCP connections for pipelined transfer. In our simulations, we assign the same window size as the last window size at the start of the idle period for the window size at the end of the idle period. The simulation in Figure 4.5 is done with one web and FTP client in the network. The simulation is run for 30.0 sec pipelined connection of HTTP/1.1 with 3 inline files. The diamond markers on the plot in Figure 4.5 give the window length at transmission time. Since we have set the maximum segment size to 1460 bytes the markers are a multiple of 1460 except for the last segment of files.

4.5.1 Simulation Results and Discussions

Simulation results for the pipelined transfer of web documents under the same traffic conditions of Section 4.1 are summarized in Table 4.6. Pipelined transfer resulted in better performance than HTTP/1.0 serial connection, because the restart of TCP connections is avoided by this modification. So, there is no waste of time with restart of TCP at every file request of same original web document.

Pipelined transfer is slightly better than the parallel connection HTTP/1.0 for the cases where parallel connection was more successful than serial connection. This still is a significant success in that pipelining requires only one connection although parallel

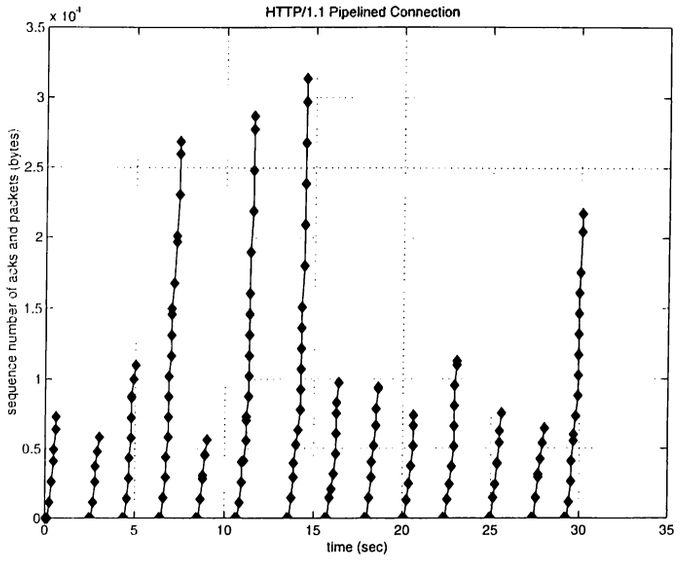

Figure 4.5: Pipelined connection in HTTP/1.1. Each web document retrieval is shown by one line connected by the sequence number at the time of transmission.

time(sec)	length(bytes)
0.17	1128 primary
0.29	1487 inline
0.40	2304 inline
0.53	2364 inline
2.61	1130 primary
2.72	2572 inline
2.85	1078 inline
2.96	1041 inline
4.48	1402 primary
4.60	7375 inline
4.85	1237 inline
4.96	1003 inline
6.51	15000 primary
7.06	1789 inline
7.19	3345 inline
7.32	6714 inline

Table 4.5: Pipelined connection sample of primary and inline file lengths.

connection requires 4 connections for a document with 3 inlines. The lowest web page retrieval rate is increased to 3.5 kbytes/sec. The drop rates are worse than parallel connection case, but it is compensated by the increased web page retrieval rate. The drop rates result from the congestion effects rather than the pipelining applied to the transfer style for the 99% utilized link.

With pipelining, the web traffic resembled the FTP file transfer. It lasts longer and if any file of large length comes from the Pareto distribution, the window size will increase. This will initiate a competition between FTP file transfer and web traffic. Hence, the highly utilized cases have large drop rates.

4.6 Segment Filled Pipelined Transfer in HTTP/1.1

Figure 4.4 shows that the window size at the time of transmission of TCP is not used in its most effective way because there is no communication between the application layer protocol HTTP and the transport layer protocol TCP. TCP can be configured such that

Table 4.6: Pipelined Connection Case - Simulation Results

# web clients	# ftp clients	percentage link utilization	effective web page retrieval(bytes/sec)	percentage packet drop rate
8	0	24%	15576.82	NONE
8	1	93%	12372.82	0.005%
8	2	96%	8460.58	0.90%
8	3	98%	7070.54	1.80%
8	4	99%	6367.26	2.37%
16	0	47%	14882.16	0.001%
16	1	85%	10890.39	0.39%
16	2	89%	8864.40	1.93%
16	3	92%	7534.22	2.85%
16	4	93%	7018.69	3.76%
32	0	87%	11760.09	0.86%
32	1	89%	8188.51	2.53%
32	2	95%	6942.78	4.46%
32	3	99%	6823.81	6.51%
32	4	96%	5573.52	7.52%
64	0	92%	5063.63	12.00%
64	1	95%	4916.73	12.99%
64	2	94%	4116.97	15.60%
64	3	95%	3886.68	16.14%
64	4	99%	3507.19	18.03%

it can wait for its window to be filled until some timeout, before continuing with the transmission. This was implemented with the Nagle algorithm [24].

During a pipelined transfer we would like to observe true pipeline such that the files are transmitted end-to-end without any break between the windows of TCP. TCP requires, most of the time, breaking the files into pieces in order to fit them into the available window size at the time of transmission. For example, for a possible window size of 512 bytes, let there be end-to-end file lengths of 250 bytes, 200 bytes, and 700 bytes, the transmission will be one packet at a time, 250 bytes, 200 bytes, 512 bytes and 188 bytes. Hence the window size will not be used fully in that case.

In this section we will introduce the results obtained through simulation of the ideal

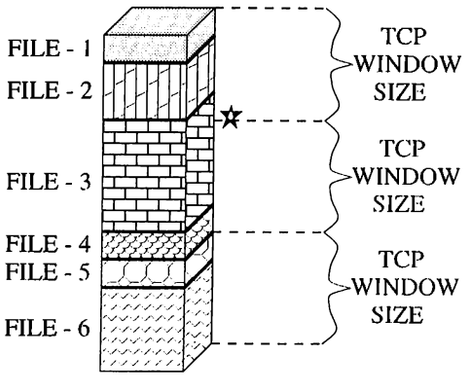

Figure 4.6: Desired pipelined transmission through TCP's window algorithm.

case: The files are broken into pieces that fit into the window size at the time of transmission. That is, the length of the files to be transmitted are chosen as multiples of the current TCP segment size. Results show desired transmission characteristics through an internet connection. The desired pipeline transmission is illustrated in Figure 4.6.

Figure 4.6 shows a typical case where the file FILE - 3 has a length that does not fit into the TCP window size at that time. Ideally, the file should be broken and sent in two pieces. This is impossible with the current transmission protocol, TCP. What actually is done with TCP is that the files are taken one by one and are sent through the segment size valid at that time. This prevents the protocol from sending data end-to-end with full efficiency. The simulation in Figure 4.7 is done for illustration purposes with one web and FTP client in the network. The simulation is run for 30.0 sec segment filled pipelined connection HTTP/1.1 with 3 inline files. The diamond markers on the plot in Figure 4.7 give the window length at transmission time. Since we have set the maximum segment size to 1460 bytes the markers are a multiple of 1460 except for the last segment of files.

The main idea of pipelined transmission is preserved in those transmissions; however, the file lengths are adjusted such that they fill exactly one segment of TCP window each time. Hence, no underfilled TCP segment is sent unless it is the last part of a document's file.

This transmission can be implemented with a completely different method defined with HTTP for this transmission. The server may parse the primary file and fetch the inline files to be transmitted with that inline. Then, it can send them together as one web packet containing all files of a web document. The web packet can be defined properly

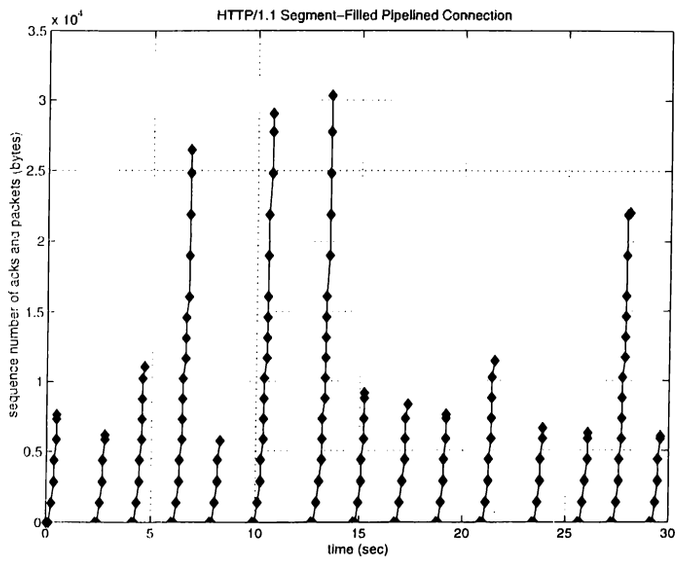

Figure 4.7: Segment filled pipelined connection in HTTP/1.1. The document transmission resembles the pipelined connection case, but sequence numbers are multiples of 1460.

time(sec)	length(bytes)
0.17	1460 primary
0.29	6155 inline
2.49	1460 primary
2.61	4691 inline
4.25	1460 primary
4.36	9615 inline
6.14	14600 primary
6.67	11848 inline
7.96	1460 primary
8.09	4265 inline
10.02	1460 primary
10.15	27575 inline
12.85	2920 primary
13.10	27401 inline
14.87	1460 primary
15.00	7666 inline

Table 4.7: Segment filled pipelined connection sample of primary and inline file lengths.

so that the client can open and fetch the files properly and display that to the user by the browser.

4.6.1 Simulation Results and Discussions

Simulations are done under the same network conditions of Section 4.1. Results are summarized in Table 4.8.

Segment filled pipelining increased the effective web page retrieval rate of 8-web-clients to approximately 21 kbytes/sec. The overall persistence of web traffic increased as seen from Table 4.8 for no-FTP-client cases. TCP segment sizes are used efficiently and savings from opening and closing of connections increased the web page retrieval rate. There are again high drop rates and instable results in the 64-web-clients cases, but these drops are mainly due to high congestion effects.

During these simulations, more filled segments of TCP are transmitted through the

Table 4.8: Segment Filled Pipelined Connection Case - Simulation Results

# web clients	# ftp clients	percentage link utilization	effective web page retrieval(bytes/sec)	percentage packet drop rate
8	0	26%	20759.15	NONE
8	1	93%	16194.68	0.08%
8	2	97%	9689.50	1.19%
8	3	98%	9522.33	1.89%
8	4	99%	7342.37	2.69%
16	0	47%	21630.72	0.05%
16	1	92%	13110.99	0.61%
16	2	97%	9715.29	2.22%
16	3	99%	8115.04	2.65%
16	4	99%	7215.52	4.13%
32	0	83%	14042.42	1.11%
32	1	98%	10270.79	3.79%
32	2	99%	7295.89	6.09%
32	3	99%	6091.82	7.39%
32	4	93%	6052.61	8.69%
64	0	96%	4447.22	16.77%
64	1	96%	4147.87	16.27%
64	2	97%	3717.14	19.15%
64	3	95%	3642.76	20.47%
64	4	99%	3537.27	21.20%

congested link than the true pipelined case. Hence, the congestion effects become severer as the number of web clients is increased to 64.

4.7 Comparison of the Outlined Methods

The performance of the 4 methods discussed above are collected in Figure 4.8 and Figure 4.9 in order to show the differences and improvements.

Both pipelined methods outperformed the parallel and serial connection HTTP/1.0. Rather than having a proportionate increase in web page retrieval rate, they have shifted

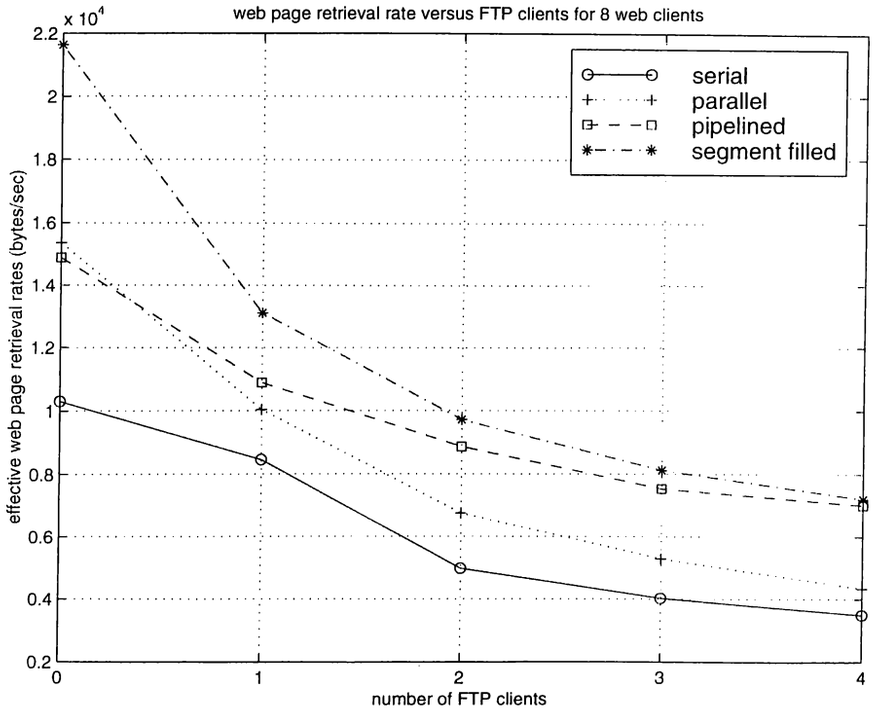

Figure 4.8: Effective web page retrieval rate versus number of FTP clients. Number of web clients is 16.

the curves of HTTP/1.0 cases up by approximately 2 kbytes/sec in web page retrieval rate.

The segment filled pipelining has much better performance than the pipelined transmission when there is no background traffic. The connections are then used with full efficiency, but with a background traffic like the case in our study, the segment filled pipelining resembles true pipelining. The background traffic we have loaded is very heavy, since only one of the FTP clients alone can utilize the link by almost 90%. These results show the worst case performance in that sense.

Parallel connection HTTP/1.0 has little better web page retrieval rate in the 16-web-clients case, as seen in Figure 4.8, than serial connection HTTP/1.0. It uses more connections at a time and since there were fewer web clients, the performance improvement was possible in that case. In the 32-web-clients simulation this is not the case, however. There is a more demanding traffic, and parallel connection could not do as well as pipelining.

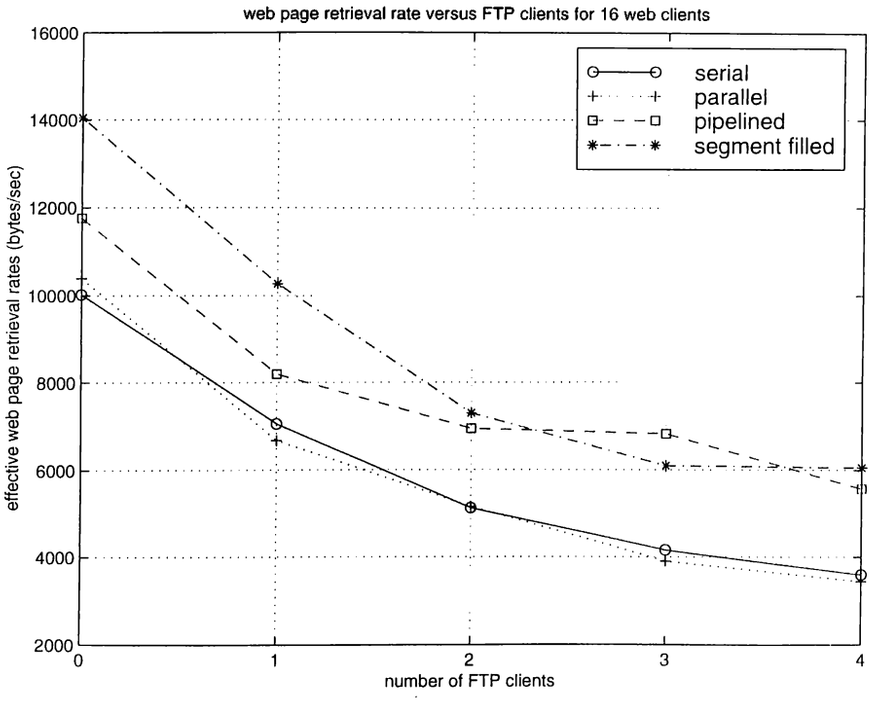

Figure 4.9: Effective web page retrieval rate versus number of FTP clients. Number of web clients is 32.

Both Figure 4.8 and Figure 4.9 illustrate that when the link is highly utilized, the methods give poor performance. For the case of 3 or 4-FTP-clients, segment filled and pipelined methods approach each other. For the same cases, parallel and serial connection performances also approach each other. Hence, the traffic conditions chosen to show the performance of the methods gave a good measure of comparison for HTTP/1.0 with HTTP/1.1.

The parallel connection HTTP/1.0 can have good performance if there are servers with capacity of opening large number of connections. In addition, the link should not be utilized in a heavy way in order to obtain performance as much as the pipelining implementation. One reason to prefer pipelining to parallel connection could be the sensitivity of the TCP connections to becoming idle for some period. If a network cannot cope with idle TCP connections for some configuration reason, then parallel connection may do the job as satisfactory as the pipelining for low utilized links.

4.8 Effects of TCP Initial Window Increase

Throughout all the simulations done for the performance analysis of modifications on HTTP/1.0, we set the TCP initial window size to 1. In this section, the experiments done with initial window size of 2, and 3 will be discussed. With the help of the increase in initial window size, HTTP documents can be sent faster than before. Since the size of the files included in an HTTP document are small when compared with large FTP files, this method is expected to give better results to some extent. Effects of initial window increase was investigated in [22] for a rough model of pipelined transmission. Here, we investigate its effects with true pipelined and segment filled pipelined method, and compare them with the parallel and serial HTTP/1.0 connections.

4.8.1 Simulation Results and Discussions

Overall, significant improvements are observed with the initial window size increase for all of the four methods outlined. But the increase in the web page retrieval rate is

Table 4.9: Initial Window Increase in HTTP/1.0 - Simulation Results

webs	ftps	effective web page retrieval rate(bytes/sec)					
		parallel HTTP/1.0			serial HTTP/1.0		
		IW=1	IW=2	IW=3	IW=1	IW=2	IW=3
8	0	15707	19271	20522	10368	12748	13082
8	1	12437	15002	16029	9159	10966	11338
8	2	7186	13599	14608	5100	5839	6350
8	3	5726	6601	8003	4087	4259	2779
16	0	15353	17969	18627	10304	12233	13131
16	1	10037	11315	11922	8452	10210	10406
16	2	6757	8337	8896	4986	6164	6436
16	3	5293	8987	8565	4027	4724	4925
32	0	10391	11984	11574	10016	12101	12602
32	1	6676	7950	8512	7051	8347	8775
32	2	5159	6122	6393	5129	5966	6479
32	3	3907	4662	5488	4156	4719	4892

less as initial window size is increased from 2 to 3 than the increase seen from 1 to 2.

This decrease in the improvement gives an idea about how things would go worse if we increase the initial window size further. Hence, [22] also suggests that we can leave the initial window size at 2 or 3 segments depending on the network conditions.

In the tables Table 4.9 and Table 4.10 the $IW = 1$ case is also included for the 4 main simulation cases for the purpose of comparison. There is a slight increase in retrieval rates for all of the cases of consideration. However, the web clients become more persistent in their demands for bandwidth from beginning of transmission with TCP initial window increase. This adds the risk of obtaining large drop rates because of congestion effects in the network.

Table 4.10: Initial Window Increase in HTTP/1.1 - Simulation Results

webs	ftps	effective web page retrieval rate(bytes/sec)					
		pipelined HTTP/1.1			segment filled HTTP/1.1		
		IW=1	IW=2	IW=3	IW=1	IW=2	IW=3
8	0	15576	17419	18152	20759	23271	25008
8	1	12372	13586	14001	16194	17651	19230
8	2	8460	11638	8956	9689	11357	12579
8	3	7070	9125	8956	9522	8623	9347
16	0	14882	16267	16778	21630	21875	23157
16	1	10890	11378	11962	13110	15504	16458
16	2	8864	8986	9172	9715	10519	14038
16	3	7534	7552	7553	8115	8573	9305
32	0	11760	13740	11946	14042	14107	15382
32	1	8188	8545	8678	10270	9670	10316
32	2	6942	7751	7766	7295	7954	9073
32	3	6823	6845	5997	6091	6206	6447

We have included plots for the initial window size increase cases. The increase in the effective web page retrieval rate is seen in the plots. But initial window increases to 3 segments sometimes results in performance degradation for highly utilized links, e.g. for 3 or 4 FTP clients. Figure 4.10 shows the initial window increase effects on retrieval rate. The upper plot is for 8 web clients and the lower is for the 16 web clients case. Initial window size increase makes a performance improvement as seen from the parallel up-shifted rate graph.

Although in the serial and parallel cases (Figure 4.10 and Figure 4.11) the improvement in the web page retrieval rate is seen in a parallel up-shifted graph, this is not the case in pipelined connection (Figure 4.12 and Figure 4.13). In the pipelined connection cases the initial window increases resulted in performance degradation in the $IW = 3$ case. Hence, further increase in the initial window size would not be that good.

The problem with initial window sizes larger than 3 is a result of congestion effects. By considering the worst case, that much persistent demand for extra bandwidth cannot be handled by the network. The worst cases for these simulations are the most utilized cases, that is, with 3-FTP-client case. With a 99% utilized link, the larger initial window size will attempt to transfer burst of data. This will increase the drop rates, hence the performance approaches the case with $IW = 1$.

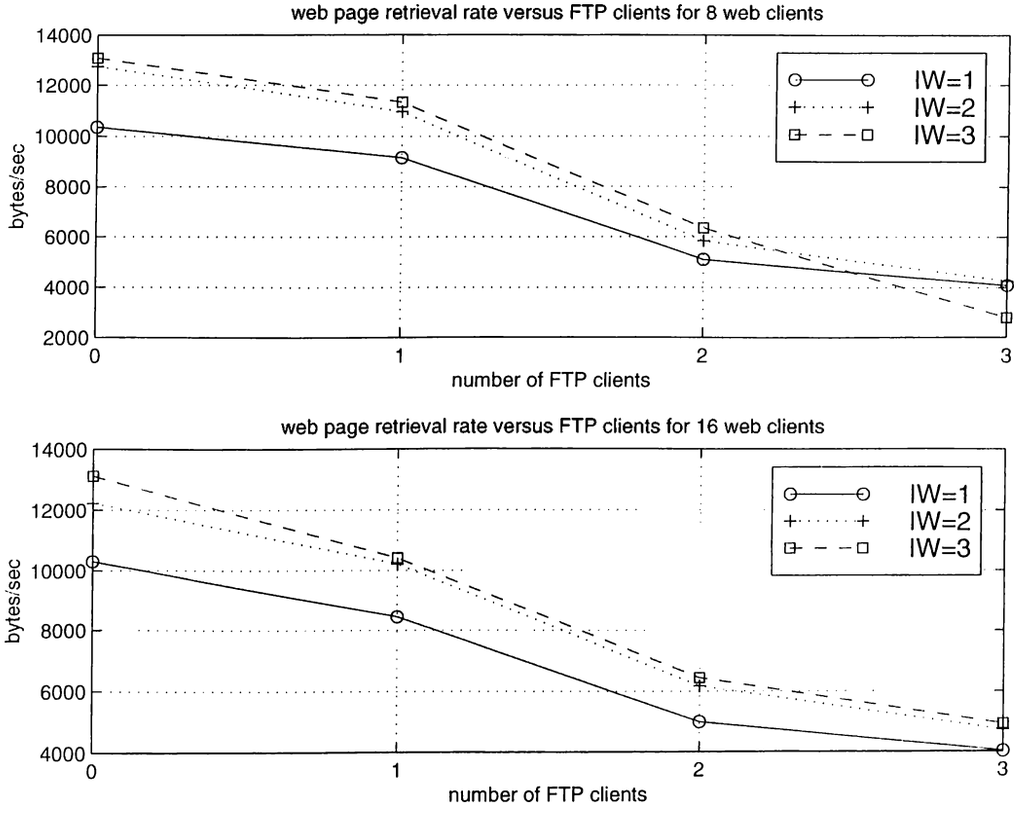

Figure 4.10: Effective web page retrieval rate versus number of FTP clients. Number of web clients is 8 and 16. Serial connection case simulation results are plotted.

The initial window size increase is a modification to TCP. Transport protocol is modified by this little modification in order to fit into transaction-oriented nature of internet.

With this modification the overhead introduced by TCP is not avoided. However, the interaction part of the connection where request response traffic takes place is faster. Here, the small length characteristics of the primary and inline files is used. The transactions most of the time take small number of request-response messages.

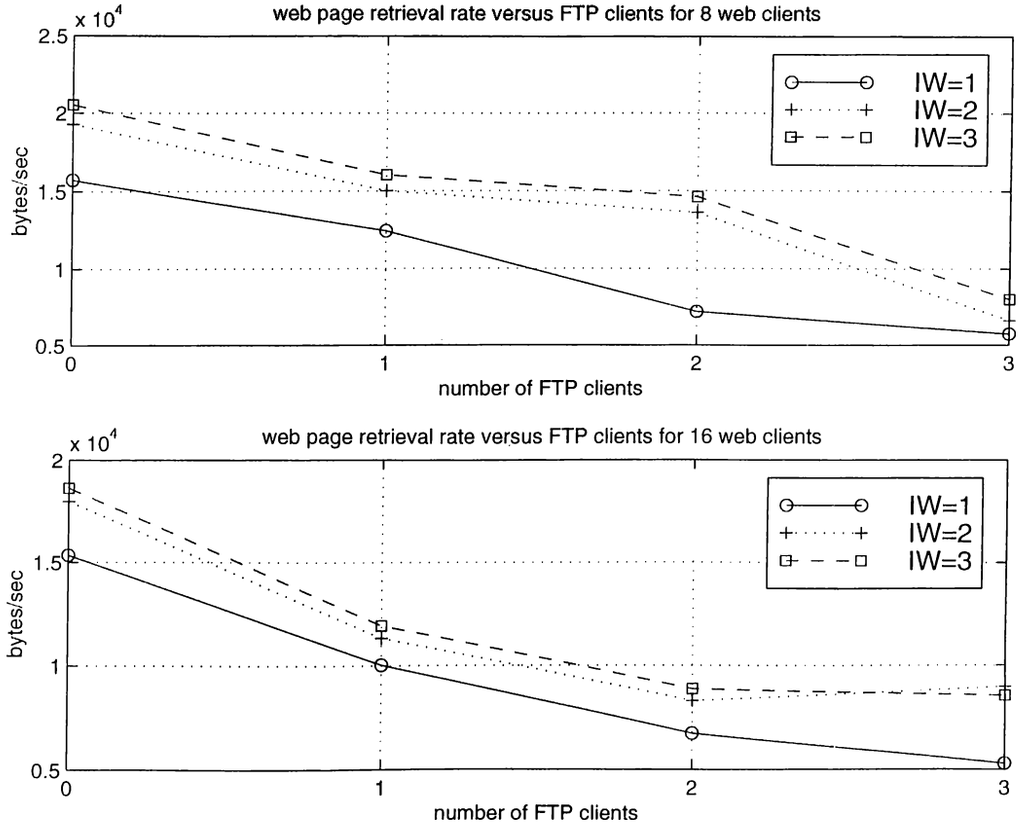

Figure 4.11: Effective web page retrieval rate versus number of FTP clients. Number of web clients is 8 and 16. Parallel connection case simulation results are plotted.

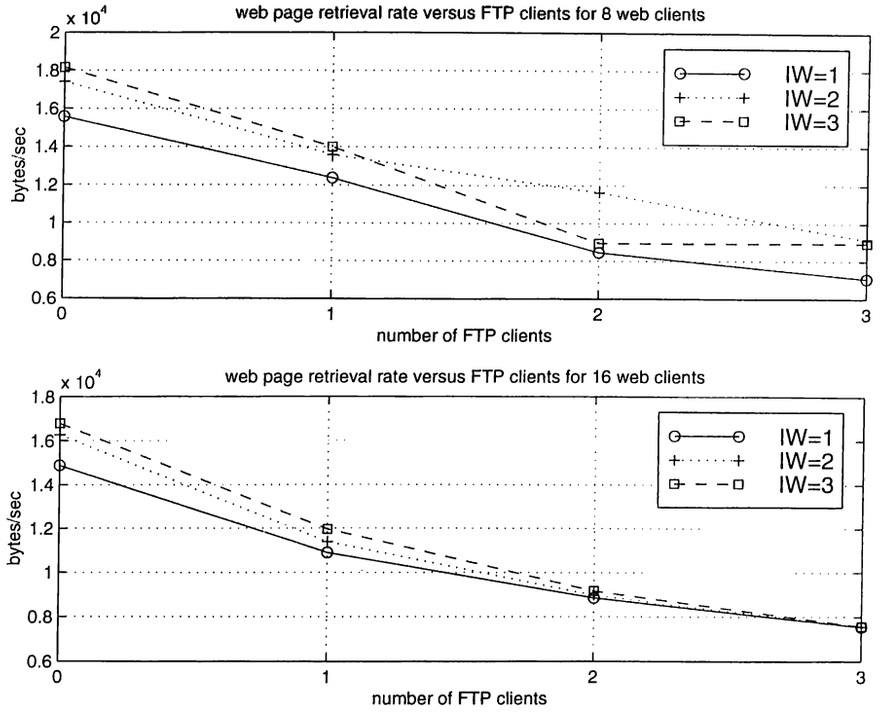

Figure 4.12: Effective web page retrieval rate versus number of FTP clients. Number of web clients is 8 and 16. Pipelined connection case simulation results are plotted.

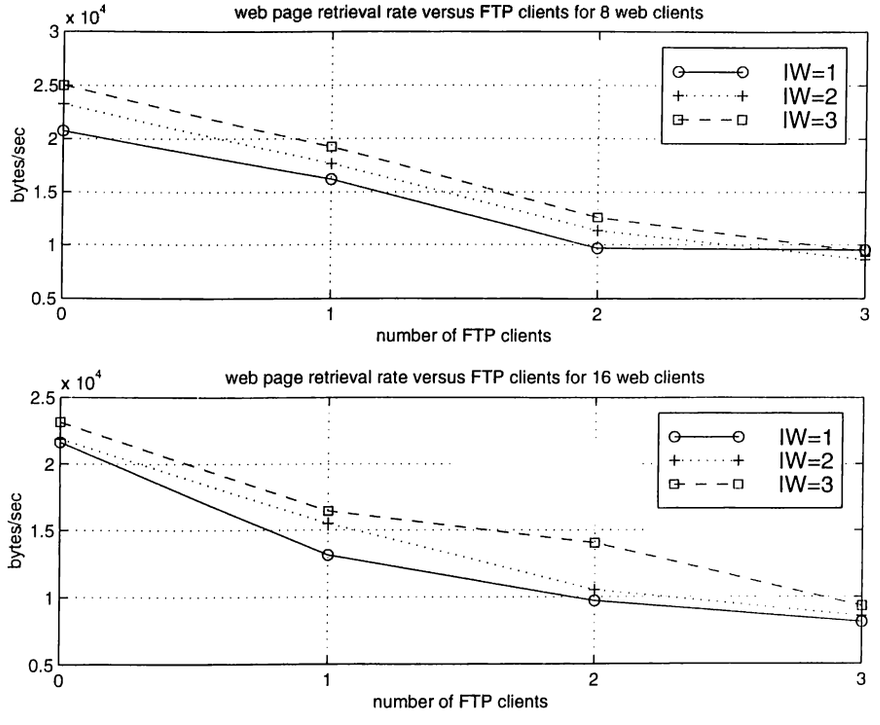

Figure 4.13: Effective web page retrieval rate versus number of FTP clients. Number of web clients is 8 and 16. Segment-filled connection case simulation results are plotted.

Chapter 5

Concluding Remarks and Future Directions

Internet applications have a quick changing nature. The protocol designed as HTTP/1.0 lacks the appropriate mechanism in order to handle the web traffic. This introduced the undesired latency problem. People prefer to retrieve their requests in relatively shorter time. Latency through web connections can be reduced since latency is not completely caused by low bandwidth links.

In this thesis we concentrated on the latency problem of HTTP resulting from its connection establishment and retrieval mechanism. In addition, we investigated its interaction with the reliable underlying protocol TCP. We proposed a new method of retrieval, the segment-filled retrieval of web documents and we have done new simulations with extensive models of HTTP/1.0 and HTTP/1.1 and web traffic. The serial and parallel connection HTTP/1.0 characteristics are compared with the true-pipelined and segment-filled connection methods of HTTP/1.1. HTTP/1.1 cases outperformed in the same traffic conditions simulated with the ones for HTTP/1.0.

Typical HTTP versions used in our internet connections have serial connection retrieval method. Hence, it is the slowest of all of the cases simulated in this thesis. It opens one connection for each of the primary and inlines. In addition to having an overhead of unnecessary open and close connection phases, it cannot reach larger window

sizes with TCP. This makes the transmission limited to a small bandwidth requirement. It does not let the congestion control algorithm of TCP demand for extra bandwidth.

The parallel connection HTTP/1.0 can only be implemented by some powerful servers in internet. Parallel connection requires the server to open many more connections at the same time than the serial case. This type of connection has the same amount of overhead in terms of open and close connection phases with TCP as in the case with serial connection. In addition, it has a large processing overhead for the server machine. The server should open approximately 3 times as many connections as for one web document with 3 inlines. This is a huge processing burden. Parallel connection also has the disadvantage of less demanding TCP connections. Since transactions did not change when compared with serial, the request-response traffic again cannot increase the window size of TCP. This results in short bursts of data transmission rather than a demand for extra bandwidth.

The transaction-oriented nature of internet traffic should be taken into consideration [19]. In this respect, pipelining method was proposed by several researchers and hence, it is included in the standard document [2]. There are again experiments to be completed in order to have pipelining method settled at its full efficiency. We have done simulations in order to monitor its performance with the models we have built by the Network Simulator.

Pipelining method gets rid of the overhead of open and close connection phases of TCP. It makes TCP approach to the transaction-oriented nature of web traffic better with this modification [25]. In addition, TCP can enlarge its window length by this approach, and link utilization is better achieved. Pipelined requests and responses through the same connection allows TCP congestion control algorithm to reach larger window sizes. Hence, demand for extra bandwidth through the link is guaranteed. Besides, the overhead of processing power for the server machine is reduced. It opens only one connection and the web document is retrieved through that connection. More than one transaction is finished through that same connection. This relaxes the server side because it will not need to store the information for e.g., 3 or 4 transactions but only one of them for one web document with 2 or 3 inline files.

Pipelining method achieved very successful results from the simulations when compared with serial and parallel connection HTTP/1.0. It was expected to perform better than the serial connection HTTP/1.0. The number of connections were reduced to one third for a typical page with 3 inlines, and the open and close connection phases are

discarded. But, it is interesting that it is also better than the parallel connection case. Retrieving the files of the web document all at once and retrieving them through the same connection gave favorable results for the pipelining. This is an achievement for pipelining method. We have opened only one connection for the whole web document with pipelining method. Hence, the latency is reduced in addition to savings in the processing overhead for the server machine. These are achieved through only one connection.

The pipelining method was proposed as a modification for the serial and parallel connections. If we take the window based transmission method of TCP, we proposed to transmit files end-to-end without any break. This enables the transmission of a full window length of data at each transfer of TCP. In the true pipelined case, files were transmitted one by one. Hence, there were unfilled windows sent through the connection. We proposed to send them without any break. The simulation results showed a clear improvement with respect to all cases simulated.

The segment-filled connection is better than true-pipelined method in its transmission method. But, implementation details may make it undesirable for the server side processing overhead. In order to send primary and inlines end-to-end, one must process on the server side the primary file. That is, the server should parse the HTML file, the primary, and decide on the inline files to be transmitted with that primary file. There can be a method called GETWEB defined as retrieval of all files of a single web document. The server which has taken that command from the client will prepare a packet with primary and inline files in it. The packet will be prepared as a sequence of files to be transmitted. The list of files and their lengths will be included in an appropriate header for the packet. This will enable the TCP regard the packet as one file and it will send it as in FTP transmission strategy. And TCP will work efficiently throughout the transmission.

One drawback of this new method may arise in cache processing on the client side. The files in the cache should not be transmitted repeatedly. Hence, the client should send the GETWEB command in an appropriate manner such that it will retrieve only the files that are not in the cache of the user. This requires some more research in the area. In addition, the processing delay should be determined during this process.

The modifications proposed in this thesis to HTTP enabled TCP to become transaction-oriented. Since internet applications are more transaction-oriented, TCP will work more

efficiently with these modifications. The transaction-oriented nature of the internet traffic is not changed. However, the traffic sensed by transport protocol becomes similar to bulk data transfer as in the case of FTP.

Bibliography

[1] W. Stallings, *Data and Computer Communications*. Prentice Hall Press, August 1996.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext transfer protocol–HTTP/1.1." Network Working Group, Request for Comments (RFC) 2068, January 1997.

[3] D. Kristol and L. Montulli, "HTTP state management mechanism." Network Working Group, Request for Comments(RFC) 2109, February 1997.

[4] B. A. Mah, "An empirical model of HTTP network traffic," in *Proceedings of INFOCOM'97*, April 1997.

[5] J. C. Mogul, "The case for persistent–connection HTTP," in *Proceedings of ACM SIGCOMM '95*, (Cambridge, MA), pp. 299–313, August 1995.

[6] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext transfer protocol–HTTP/1.0." Network Working Group, Request for Comments (RFC) 1945, May 1996.

[7] J. Touch, "TCP control block interdependence." Internet Engineering Task Force(IETF), TCP Implementers Working Group, Internet Draft, December 1996. working draft.

[8] T. V. Lakshman and U. Madhow, "Performance analysis of window-based flow control using TCP/IP: the effect of high bandwidth-delay products and random loss," *IEEE Transactions on Networking*, June 1997.

[9] W. Stevens, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithm." Network Working Group, Request for Comments(RFC) 2001, January 1997.

[10] D. D. Clark, "Window and acknowledgment strategy in TCP." Network Working Group, Request for Comments(RFC) 813, July 1982.

[11] T. J. Shepard, "TCP packet trace analysis," tech. rep., MIT, Laboratory for Computer Science, TR-494, February 1991.

[12] V. Paxson, "Growth trends in wide-area TCP connections," *IEEE Network*, 1994.

[13] S. E. Spero, "Analysis of HTTP performance problems." work in progress in USC/Information Sciences Institute, July 1994.

[14] C. R. Cunha, A. Bestavros, and M. E. Crovella, "Characteristics of WWW client-based traces," tech. rep., Boston University, Computer Science Department, 1995.

[15] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley, "Network performance effects of HTTP/1.1, CSS1, and PNG." NOTE, June 1997.

[16] V. N. Padmanabhan and J. C. Mogul, "Improving HTTP latency," in *Proceedings of the 2nd International WWW Conference*, October 1994.

[17] J. Touch, J. Heidemann, and K. Obraczka, "Analysis of HTTP performance." work in progress in USC/Information Sciences Institute, August 1996.

[18] J. Heidemann, K. Obraczka, and J. Touch, "Modeling the performance of HTTP over several transport protocols." DRAFT, June 1997.

[19] R. Braden, "Extending TCP for transactions – concepts." Network Working Group, Request for Comments(RFC) 1379, November 1992.

[20] R. Braden, "T/TCP – TCP extensions for transactions functional specification." Network Working Group, Request for Comments(RFC) 1644, July 1994.

[21] V. Jacobson, "Congestion avoidance and control," in *Proceedings of SIGCOMM'88 Symposium on Communications Architectures and Protocols*, (Stanford, CA), pp. 314–329, August 1988.

[22] K. Poduri and K. Nichols, "Simulation studies of increased initial TCP window size." Internet Engineering Task Force(IETF), TCP Implementers Working Group, Internet Draft, February 1998. working draft.

[23] V. Visweswaraiah and J. Heidemann, "Improving restart of idle TCP connections." Draft-Currently submitted for publication, August 1997.

[24] J. Nagle, "Congestion control in IP/TCP internetworks." Network Working Group, Request for Comments(RFC) 896, January 1984.

[25] D. D. Clark, "Modularity and efficiency in protocol implementation." Network Working Group, Request for Comments(RFC) 817, July 1982.