

PERFORMANCE ANALYSIS OF PERSISTENT HTTP OVER SATELLITE
LINKS

A Thesis Presented To

The Faculty of the

Fritz J. and Dolores H. Russ
College of Engineering and Technology

Ohio University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Xin Chen

November, 1997

THIS THESIS ENTITLED
“PERFORMANCE ANALYSIS OF PERSISTENT HTTP OVER SATELLITE
LINKS”

by Xin Chen

has been approved

for the School of Electrical Engineering and Computer Science
and the Russ College of Engineering and Technology

Shawn D. Ostermann
Assistant Professor of School of Electrical Engineering and Computer Science

Warren K. Wray, Dean
Fritz J. and Dolores H. Russ
College of Engineering and Technology

ACKNOWLEDGMENTS

I'd like to thank Dr. Shawn Ostermann, who guided me into the magic Internetworking world. I thank Mark and all the other Internetworking Research Group members, whose discussion helps me to further my research. I thank Dr. Douglas Lawrence and the School of Electrical Engineering and Computer Science for the help during my study. I thank my parents, family, friends, for their love and support during these years. I especially thank my wife, for her love and encouragement.

DISCARD THIS PAGE

TABLE OF CONTENTS

	Page
LIST OF TABLES	iv
LIST OF FIGURES	v
1. INTRODUCTION	1
1.1 Motivation for this Thesis	1
1.2 Organization of this Thesis	1
2. HTTP BACKGROUND	3
2.1 Computer Communications Architecture	3
2.2 HTTP	4
2.3 TCP	6
2.3.1 Segments and Sequence Numbers	6
2.3.2 Reliability	6
2.3.3 Sliding Window	7
2.3.4 Nagle Algorithm	8
2.3.5 Congestion	9
2.4 HTTP Performance Inefficiencies Analysis	10
2.4.1 Inefficiencies Caused by TCP Connection Overhead	10
2.4.2 Inefficiencies Caused by Congestion Control	11
2.5 Persistent HTTP (HTTP/1.1)	12
2.5.1 Sending Requests	12
2.5.2 Closing the Connection	13
3. PERFORMANCE TESTS AND ANALYSES	15
3.1 Performance Test Environment	15
3.1.1 Hardware Setup	15
3.1.2 Software Setup	16

	Page
3.1.3 Web Page Setup	16
3.2 Test Results	16
3.2.1 HTTP/1.1 vs. HTTP/1.0	17
3.2.2 HTTP/1.1 Pipeline vs. HTTP/1.0	19
3.3 Influence of Web Page Constitution	21
3.4 Influence of Nagle Algorithm	24
3.4.1 When Can TCP Send a Data Segment	24
3.4.2 Why Disable the Nagle Algorithm	24
3.4.3 Comparisons: Disable/Enable Nagle Algorithm	25
3.4.4 Suggestions to Solve this Problem	26
4. CONCLUSIONS	32
BIBLIOGRAPHY	33
APPENDIX	35
A. READ HTTP AND TCP GRAPHS	35
A.1 How to Read an HTTP Transaction Graph	35
A.2 How to Read a TCP Time Sequence Graph	35
B. WEB PAGES	37
ABSTRACT	41

LIST OF TABLES

Table	Page
3.1 Web Page Setup	17
3.2 General Performance Comparison for CNN Web Page	17
3.3 Performance Comparisons against CNN Web Page	22
3.4 Performance Comparisons against Microsoft Web Page	22
3.5 Performance Comparisons against Mini Web Page	23
3.6 Performance of HTTP/1.1 Pipeline over HTTP/1.0	23
3.7 Performance Comparisons: Disable/Enable Nagle Algorithm	27

LIST OF FIGURES

Figure	Page
2.1 TCP/IP Layering Model	4
2.2 Simple Positive Acknowledgment with Retransmission	7
2.3 An Example of Three Packets Transmitted Using a Sliding Window	8
2.4 HTTP Transaction for a Single Request	10
2.5 HTTP/1.1 and HTTP/1.1 Pipeline	13
3.1 Hardware Setup	15
3.2 HTTP/1.1 Transaction Graph	18
3.3 HTTP/1.1 Time Sequence Graph	19
3.4 HTTP/1.1 Pipeline Transaction Graph	20
3.5 HTTP/1.1 Pipeline Time Sequence Graph	21
3.6 HTTP/1.1 Time Sequence Graph (enable nagle)	26
3.7 HTTP/1.1 Pipeline Time Sequence Graph (enable nagle)	27
3.8 Disable Nagle Algorithm after the Final Response	28
3.9 Server Closes HTTP/1.1 Pipeline Connection First	30
3.10 Server/Client Closes the Connection	30
B.1 Mini Home Page	37
B.2 CNN Home Page (part 1)	38

Figure	Page
B.3 CNN Home Page (part 2)	38
B.4 CNN Home Page (part 3)	39
B.5 CNN Home Page (part 4)	39
B.6 Microsoft Home Page (part 1)	40
B.7 Microsoft Home Page (part 2)	40

1. INTRODUCTION

1.1 Motivation for this Thesis

The World Wide Web (WWW) is very popular today because it offers a convenient method to access distributed multimedia information in an integrated environment with a general, user friendly interface. The introduction of the WWW promoted the development of the Internet. Meanwhile, WWW occupies a higher and higher proportion in the total Internet traffic [Gra96]. However, the Hypertext Transfer Protocol version 1.0 (HTTP/1.0) [BLFF96], the current WWW protocol, does not work very efficiently. It introduces extra overhead on the network, wastes the network's capacity, and more importantly, takes a long time to fetch all the information that the user requires. This is especially obvious in long delay, high bandwidth satellite links.

Several proposals have been suggested to improve HTTP performance. These proposals include Persistent HTTP (HTTP/1.1) [FGM⁺97], Transaction TCP [Bra94] and Sharing TCP Control Blocks [Tou97]. This thesis investigates the performance of a particular Persistent HTTP implementation over satellite links.

1.2 Organization of this Thesis

This thesis contains four chapters. Chapter one discusses the motivation for this work. Chapter two analyses the performance problems of HTTP/1.0 over TCP, introduces the idea of Persistent HTTP (HTTP/1.1) and Persistent HTTP with Pipeline (HTTP/1.1 Pipeline). Chapter three compares HTTP/1.0, HTTP/1.1 and

HTTP/1.1 Pipeline performance over satellite links, analyses the influence of the web page constitution on HTTP performance, and discusses the influence of the nagle algorithm on HTTP performance. Chapter four addresses the conclusions from the comparisons and analyses.

2. HTTP BACKGROUND

2.1 Computer Communications Architecture

A simple computer communication system consists of a sender, a receiver and a communication channel. The sender and the receiver use a set of rules or conventions called a *protocol* to govern the information exchange behaviors between them.

In order to make the computer communication system function properly in various communication environments, the protocols are organized in four relatively independent layers [Com95], as shown in Figure 2.1:

- Network Interface Layer.
- Internet Layer.
- Transport Layer.
- Application layer.

The network interface layer is concerned with the data exchange between the computer and the network it attaches to. The Internet layer handles the communication from one computer to another. The transport layer guarantees the reliability of data exchange between the application programs. The application layer contains the logical rules needed to support the various user applications.

Normally, at each protocol layer, the protocol data unit is divided into two parts: the header and the data. The header portion contains the control information used by the peer protocol. A higher level protocol is encapsulated into the data portion of a lower level protocol.

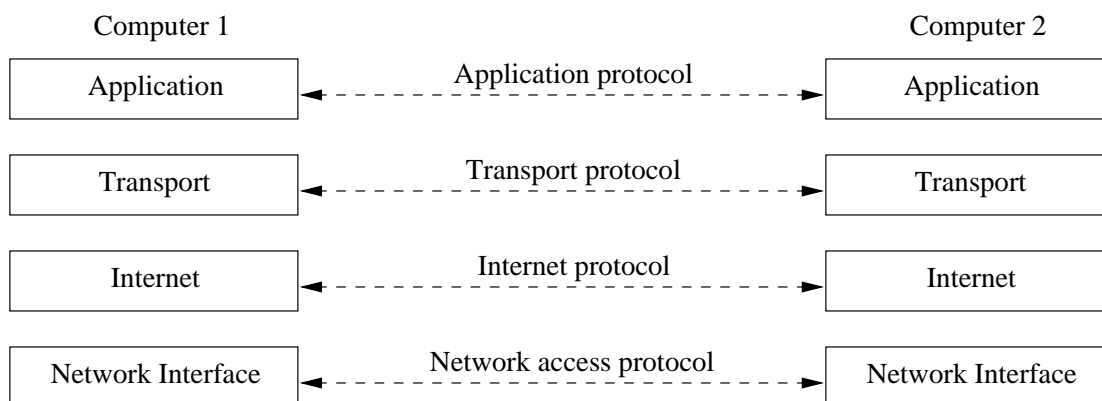


Figure 2.1 TCP/IP Layering Model

2.2 HTTP

The Hypertext Transfer Protocol (HTTP) is an application-level protocol. It is layered above the reliable transport-level protocol, TCP. The HTTP protocol is based on a request/response paradigm. A typical successful HTTP communication works like this: the client establishes a connection with the server and sends a request to the server, the server parses the request, sends back the response, and then closes the connection. The client can only send *one* request and get *one* response per connection. (Some implementations of HTTP/1.0 allow multiple requests per connection, though this scheme is not documented in RFC 1945 [BLFF96]. This thesis only discusses the default HTTP/1.0 behavior.)

An HTTP request includes: a request method, a Uniform Resource Location (URL) which indicates the resource requested, the protocol version, and other HTTP control information. An HTTP response includes: protocol version, success or error code, server information, entity meta-information containing content length, and body content [BLFF96].

HTTP works in client/server mode [Com95]. The HTTP server has a lot of information sources, for example, text files, images, software. The HTTP client (normally called a browser) fetches these information sources from the server according to user's instructions. Netscape Navigator [Cor97b] and Internet Explorer [Cor97a] are two popular browsers.

A URL may refer to various kinds of network data objects or services, but the primary type of object requested is Hypertext Markup Language (HTML) [BLC95]. HTML is a simple data format used to create hypertext documents. It supports inline images, that is, in the HTML source file, it uses URLs to address the locations of optional inline images; the browser should fetch the inline images as well as the HTML source file.

Suppose we have a HTML file called `hello.html` which contains two inline images `hello1.gif` and `hello2.gif`:

```
Hello World!  
<IMG SRC=image1.gif>  
<IMG SRC=image2.gif>  
Goodbye!
```

In this case, the HTTP transaction will work like this:

1. The browser establishes a connection with the server and sends a request for `hello.html`.
2. The server sends back the content of `hello.html` in the corresponding response and closes the connection.
3. The browser parses `hello.html`, finds the information about `image1.gif` and `image2.gif`.
4. The browser establishes two new connections with the server and requests for `image1.gif` and `image2.gif` respectively.

5. The server sends back `image1.gif` and `image2.gif` and closes both connections.

In this example, the browser will use three connections to fetch one HTML file and two inline images. In the example above, note that steps 3–5 can be done simultaneously with steps 1–2, as Netscape does, to save time.

2.3 TCP

TCP offers HTTP a reliable, stream-oriented, sliding window based, full-duplex connection [Com95]. Both sides of the connection have a send buffer and receive buffer to cache the data stream temporarily.

2.3.1 Segments and Sequence Numbers

TCP is stream oriented, it views the data as a sequence of 8-bit octets. In order to identify each octet, TCP assigns it a number in sequence which is unique during one connection, called *sequence number*.

When transferring data, TCP divides the sequence of octets into segments. Normally, each segment travels across the network independently in one IP datagram. By default, TCP connections over two Ethernet-connected hosts have a *Maximum Segment Size (MSS)* of 1460 bytes, connections over wide area network will use an MSS of 512 bytes or 536 bytes.

2.3.2 Reliability

In order to guarantee the reliable stream delivery service, a fundamental technique called *positive acknowledgment with retransmission* is used. The receiver is required to send back an *acknowledgment* message which tells the sender the next octet expected after it receives a certain segment. The sender also starts a timer when it sends a

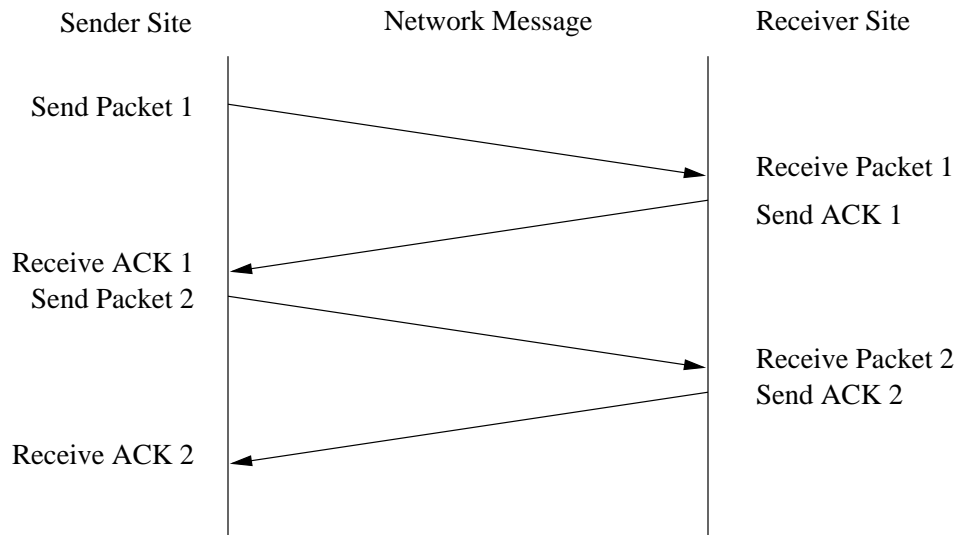


Figure 2.2 Simple Positive Acknowledgment with Retransmission

segment. If this timer expires before the acknowledgment is received, the sender will retransmit the segment.

2.3.3 Sliding Window

In the simple positive acknowledgment protocol, the sender waits for the acknowledgment before sending another segment, as shown in Figure 2.2. It does not use the network capacity efficiently. This will produce poor performance in a high delay communication environment, for example, the satellite communication system, so that a scheme called *sliding window* is introduced.

A sliding window protocol has a window whose size may be fixed or variable. The protocol may transmit all the packets inside the window before receiving any acknowledgments, as shown in Figure 2.3. A packet that has been sent but not acknowledged is called *unacknowledged*. When the sender receives an acknowledgment, it will forward the sliding window to the octet the acknowledgment indicates.

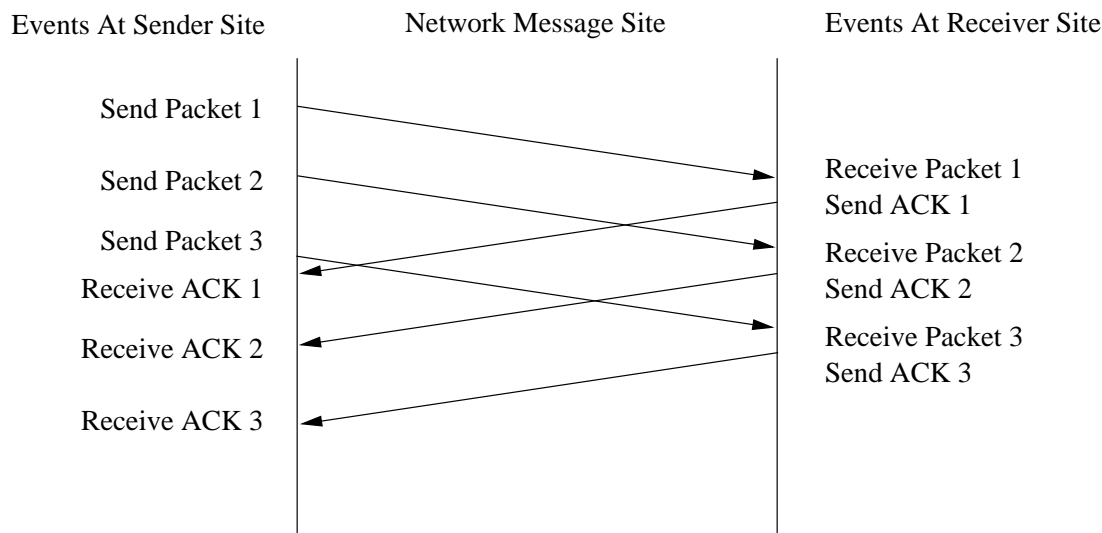


Figure 2.3 An Example of Three Packets Transmitted Using a Sliding Window

TCP uses a variable-sized sliding window. Besides indicating the next octet expected, the acknowledgment also provides a window advertisement message, which tells the sender how many additional octets the receiver is prepared to receive, called the *receive buffer size*. To make the sender and the receiver work cooperatively, the send side's sliding window size should not exceed this receive buffer size.

2.3.4 Nagle Algorithm

If each acknowledgment advertises a small amount of space available and each TCP segment carries a small amount of data, the network utilization will be reduced because it will take more TCP segments to transfer the same amount of data and in each TCP segment, the ratio of data size to header size is small. This problem is called *silly window syndrome (SWS)*. The nagle algorithm is a send-side silly window avoidance algorithm. According to [Com95], its main idea is:

When an application program wants to send data over TCP, if the TCP has data which has been sent but not acknowledged, put this new data

into the output buffer, wait until it accumulates to fill a full size segment and send that segment. If it is still waiting when the next acknowledgment arrives, send all the data in the buffer.

2.3.5 Congestion

TCP packets are routed and dispatched by a series of routers from the source to the destination. Each router has a finite queue to store the incoming packets. If the number of incoming packets exceeds the limit, the router will discard some packets. In this case, network congestion occurs.

The Internet traffic changes capriciously. In order to work efficiently without causing the network to collapse, TCP uses a second limit, called the *congestion window*. At any time, the send side's sliding window size is limited by Equation 2.1.

$$\textit{Allowed_window} = \textit{min}(\textit{receiver_advertisement}, \textit{congestion_window}) \quad (2.1)$$

If the TCP connection is in steady status and not congested, the size of *congestion_window* is equal to the size of *receiver_advertisement*. TCP uses *slow-start*, *congestion avoidance* to determine the size of *congestion_window*.

When TCP starts to inject traffic into a new connection or has suffered a period of congestion, slow-start will be adopted. It initializes the congestion window to one single segment, and increases it by one segment when an acknowledgment arrives. In this way, it can increase the congestion window exponentially.

When the congestion window size reaches one half of the previous size before congestion, the connection enters the congestion avoidance period. The congestion window size is increased by one segment only after *all* the segments in the window have been acknowledged.

2.4 HTTP Performance Inefficiencies Analysis

We can use two criteria to evaluate the performance of HTTP. One is the total elapsed time from the time that the browser requests the base HTML file to the time that the browser gets the HTML file as well as all the possible inline images (we call this a *web page*). The other criteria is the total packets used to complete that work.

2.4.1 Inefficiencies Caused by TCP Connection Overhead

HTTP/1.0 handles one HTTP request per TCP connection, while TCP uses three-way handshake to establish and close a connection [Com95]. A typical HTTP transaction for a single request is shown as Figure 2.4.

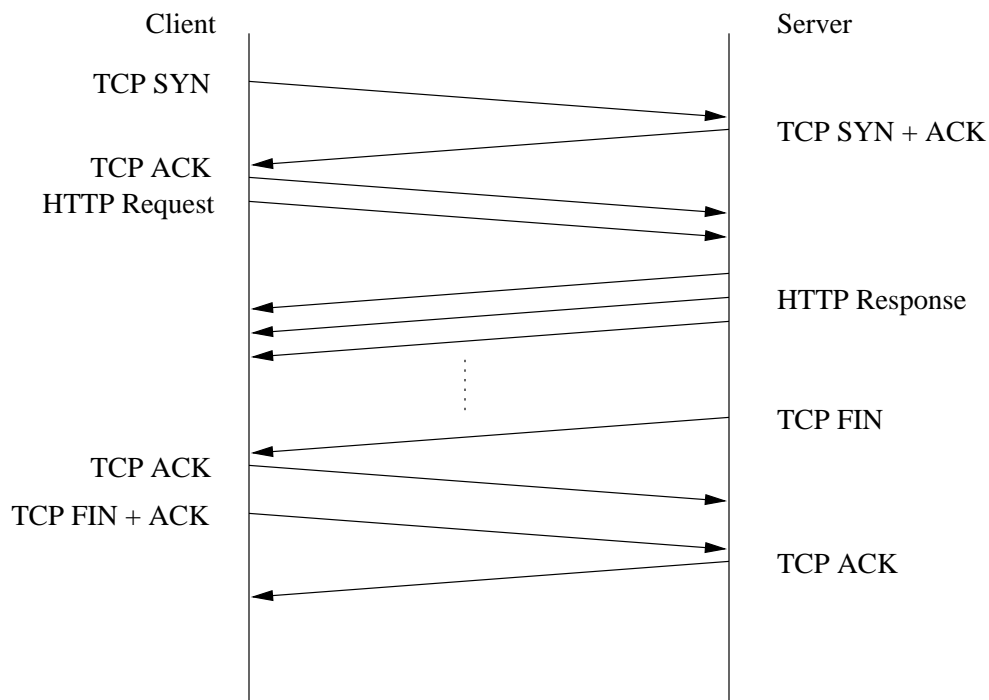


Figure 2.4 HTTP Transaction for a Single Request

It is clear that the user will wait at least 2 *round-trip times (RTTs)* (1 RTT is used for setting up TCP connection) to get the first HTTP data packet from the server after the browser decides to fetch the URL, and more than one RTT to close the TCP connection. This is true for the base HTML file as well as the inline images. It is common to see an HTML file containing more than ten different inline images in today's Internet. For example, if the user wants to fetch an HTML document with 10 inline images, the browser needs 11 TCP connections. Besides the time used for transferring real HTTP data, the user has to spend extra time to wait to establish and close the TCP connections. One request per connection also introduces at least 7 extra packets per connection to an HTTP request/response to establish and close the TCP connection.

Many of these images are icons with small size. It takes only a few segments to transfer them. In that case, the influence of extra waiting time and packets that caused by establishing and closing TCP connection is significant.

2.4.2 Inefficiencies Caused by Congestion Control

In response to congestion control, each TCP connection follows the rules of slow-start and congestion avoidance [Com95]. This also introduces HTTP performance inefficiencies.

For example, both the HTTP server and client have a 16KB TCP send buffer and receive buffer, the MSS is 512 bytes. The server is required to send a 20KB file to the client. It takes $20KB/512B = 40$ segments to hold the data according to Equation 2.2.

$$segment\ number = \frac{data\ size}{MSS} \quad (2.2)$$

In steady state, the sliding window size is 16KB (see Equation 2.1), i.e., 32 segments. In this case, TCP can send the first 32 segments just in one RTT followed by the remaining 8 segments in the second RTT, for a total of 2 RTTs.

If this occurs in a new connection, however, we must follow the rule of slow-start and congestion avoidance. Suppose that TCP knows that the window size before congestion is 16KB, so that it knows that after half of 16KB, i.e., 8 KB (16 segments), it will enter the congestion avoidance phase from the slow-start phase. In this case, it takes $1 + \log_2 16 = 5$ RTTs to reach the threshold. During the slow-start phase, TCP sends out total $2^5 - 1 = 31$ segments, then it takes another 1 RTT to send the remaining 9 segments, for a total of 6 RTTs. Note that this happens in every TCP connection which carries one HTTP request.

2.5 Persistent HTTP (HTTP/1.1)

There are several ways to improve HTTP performance. Netscape allows several parallel TCP connections, each of which carries a single request/response. This can shorten the perceived waiting time significantly. Persistent HTTP (HTTP/1.1) attempts to avoid the TCP connection overhead and the slow-start overhead introduced by one request/response per connection HTTP protocol by allowing multiple requests/responses per connection, that is, instead of closing the TCP connection after each request/response, keep the connection alive and use it to handle multiple request/response transactions [Mog95].

2.5.1 Sending Requests

In HTTP/1.1, the client will send a new request only after it receives the response for the previous request. HTTP/1.1 Pipeline allows the client to send the request as soon as possible instead of waiting. The server should answer the requests by the same order that the client requests [NGS⁺97]. Figure 2.5 illustrates the difference between HTTP/1.1 and HTTP/1.1 Pipeline.

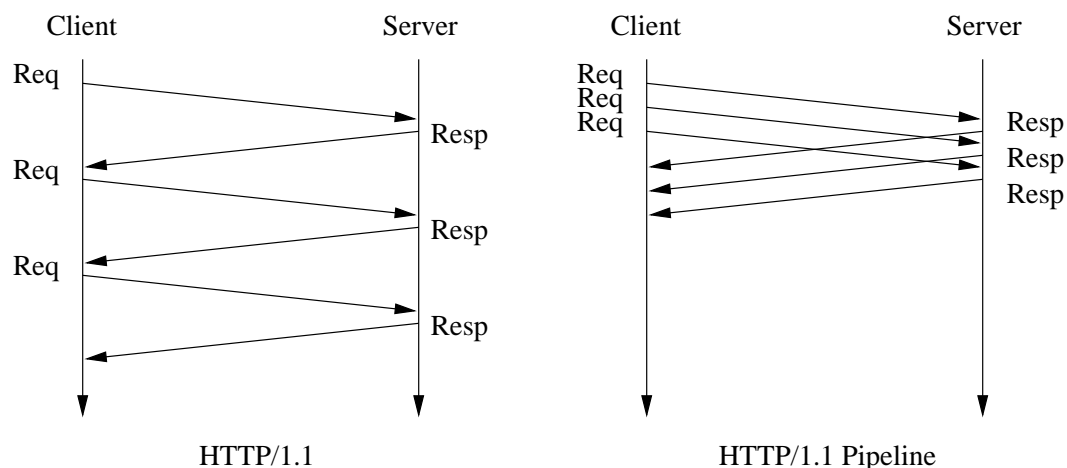


Figure 2.5 HTTP/1.1 and HTTP/1.1 Pipeline

2.5.2 Closing the Connection

TCP is a full-duplex connection. Each side can close the connection first, that is, closes half of the connection and sends a FIN to the peer. When the peer receives that FIN, it will close the other half of the connection and send a FIN back.

In both HTTP/1.0 and HTTP/1.1, there are some abnormal reasons that can cause the TCP connection to be closed. For example, the user terminates the connection arbitrarily by clicking the `stop` icon in the browser. This thesis only discusses the normal cases.

In HTTP/1.0, since one connection contains only one request, the server will close the TCP connection first after it completes the response. In HTTP/1.1, the connection is kept alive. From the aspect of the server, there might be a request at any time so that the server itself does not know when to close the connection. In the current implementation of HTTP/1.1, it is the client who closes the connection first. After the client gets and parses the HTML file, it knows how many additional requests it needs. After it sends all the requests to the server and gets all the responses, the

client will close the connection first. After the server gets the FIN, it will close the other half of the connection.

HTTP/1.1 offers additional mechanisms to decide when to close the connection. The server starts a keep-alive timer for each TCP connection, it will be reset when a new request comes. If this timer expires (no new request for a long time), the server will close the connection after it completes all the remaining requests. The server also sets an upper limit for the *requests allowed per keep-alive connection*, the server will respond to as many as that number of requests before closing the connection in a HTTP/1.1 transaction. The HTTP/1.1 protocol defines a “Connection: Close” message which can be part of the HTTP header. When the server sends that message to the client, the client must close the connection right away; when the client sends that message to the server with a request for a URL, the server will close the connection after it completes the response for that request [FGM⁺97].

3. PERFORMANCE TESTS AND ANALYSES

3.1 Performance Test Environment

This section introduces the hardware, software and web page configurations for the performance tests.

3.1.1 Hardware Setup

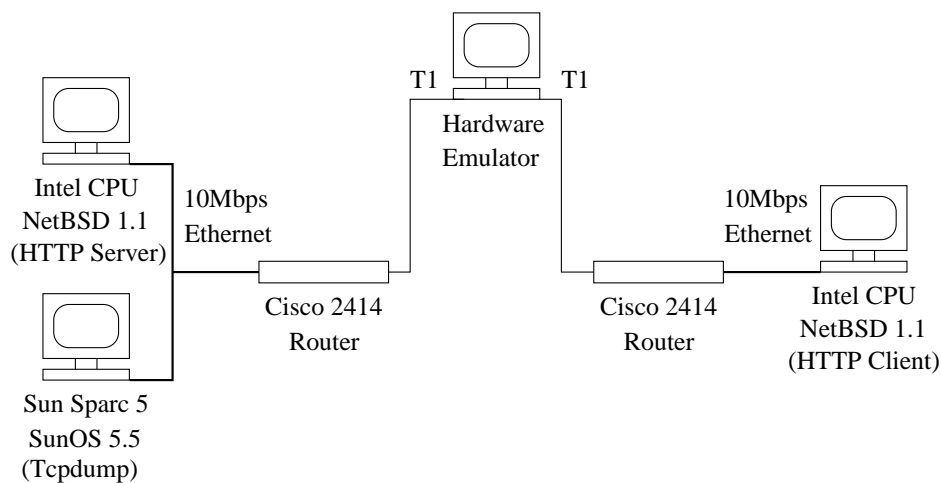


Figure 3.1 Hardware Setup

Figure 3.1 shows the hardware setup for the tests. The HTTP server and client run on Intel CPU based computers running NetBSD 1.1. The SunOS computer is used to monitor the HTTP traffic. It uses tcpcdump [dum94] to download the source traffic data for further study. The emulator is the “Data Link Simulator” made by

Testlink Corporation. It can be configured to limit the bandwidth between routers. In these experiments, it is set to 1,536,000 bps (T1). It can also model a given amount of propagation delay. For these tests, it is set to 290 ms in each direction, which is common in satellite communication.

3.1.2 Software Setup

In the tests, we use the `Apache 1.2.4 HTTP server` [Apa97], the HTTP/1.0 client uses `libwww robot 4.0D1`, the HTTP/1.1 and HTTP/1.1 Pipeline client uses `libwww robot 5.1b` with different command options [Nie97]. This thesis uses `tcpdump` to collect the source traffic data between HTTP client and server; it uses `tcptrace` [Ost97] to analyze the source traffic data; it uses `xplot` [She97] to study the HTTP transaction graph and TCP time sequence graph.

3.1.3 Web Page Setup

To study the influence of the constitution of web pages over HTTP, HTTP/1.1 and HTTP/1.1 Pipeline, this thesis investigates three web pages, two from popular Internet home pages: Microsoft Home Page, CNN [Inc97] Home Page, and a locally created web page called “Mini Home Page” [AK97] which contains many images with small size. Table 3.1 shows the constitutions of these three web pages. Microsoft Home Page has many small sized images. In CNN Home Page, medium sized images make up the majority. The contents of these web pages are shown in Appendix B.

3.2 Test Results

The tests use CNN as the web page, and run over a quiet network. They test the performance of HTTP/1.0, HTTP/1.1 and HTTP/1.1 Pipeline. The results are listed in Table 3.2.

Table 3.1 Web Page Setup

Web Page	Text Size (Bytes)	Number of Images vs. Image Size				Total Size (Bytes)
		<1KB	1-4KB	4-8K	>8K	
CNN	27,698	2	2	8	3	205,950
Microsoft	25,987	12	4	1	2	68,343
Mini	1,481	7	20	0	0	28,709

Table 3.2 General Performance Comparison for CNN Web Page

	HTTP/1.0	HTTP/1.1	HTTP/1.1 Pipeline
Max parallel TCP connections	6	1	1
TCP connections used	16	1	1
Total packets	480	377	360
Data packets (client to server)	32	32	10
Data packets (server to client)	246	266	301
Total elapsed time (secs)	15.3	17.5	10.0

3.2.1 HTTP/1.1 vs. HTTP/1.0

As shown in Table 3.2, HTTP/1.0 uses more network resources (TCP connections and packets) to complete the same task. The number of data packets are close to each other, but it costs HTTP/1.0 much more TCP setup/close packets. On the other hand, HTTP/1.1 spends more time than HTTP/1.0 to finish the job. In the HTTP/1.1 protocol, the client submits one request after it receives the response for the previous request, as illustrated in Figure 3.2. (Appendix A shows how to read HTTP and TCP graphs.)

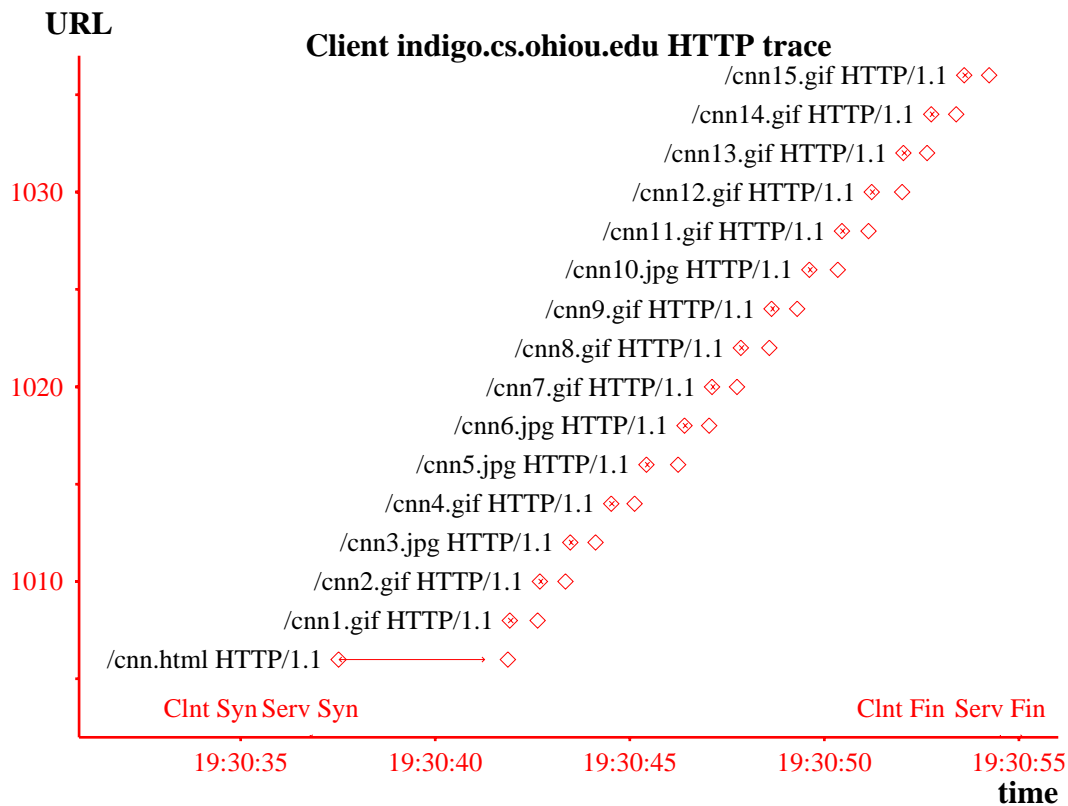


Figure 3.2 HTTP/1.1 Transaction Graph

In Figure 3.3, the corresponding TCP time sequence graph for this transaction (from server to client, dump the tcp traffic data at the server side), it shows that after the slow-start period, the communication channel is not used efficiently. One important reason is that although the receive window is large (16KB), the sizes of the inline images are small (normally, less than 16KB), the server will complete the response for one request in a few segments and wait for the client to send another request.

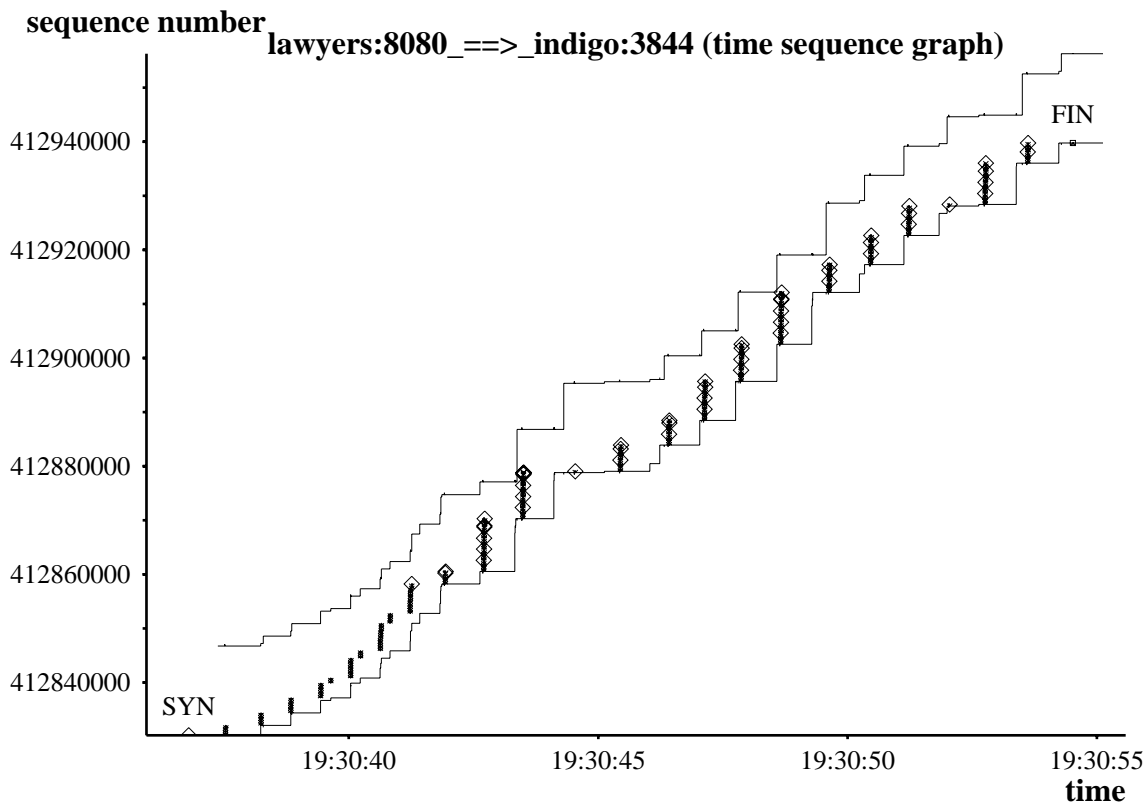


Figure 3.3 HTTP/1.1 Time Sequence Graph

3.2.2 HTTP/1.1 Pipeline vs. HTTP/1.0

The pipelining technique is introduced to improve HTTP/1.1 performance. As shown in Figure 3.4, the client sends as many requests as possible instead of waiting for the response of the previous request.

Compared to HTTP/1.0 and HTTP/1.1, HTTP/1.1 Pipeline reduces the elapsed time significantly. It also reduces the total number of packets compared to HTTP/1.0. It uses fewer data packets from the client to the server because it can encapsulate several requests into one segment. However, it uses more data packets to transfer the same response from the server to the client than HTTP/1.0 and HTTP/1.1 (see Table 3.2). In HTTP/1.0, the server closes the connection after a single response,

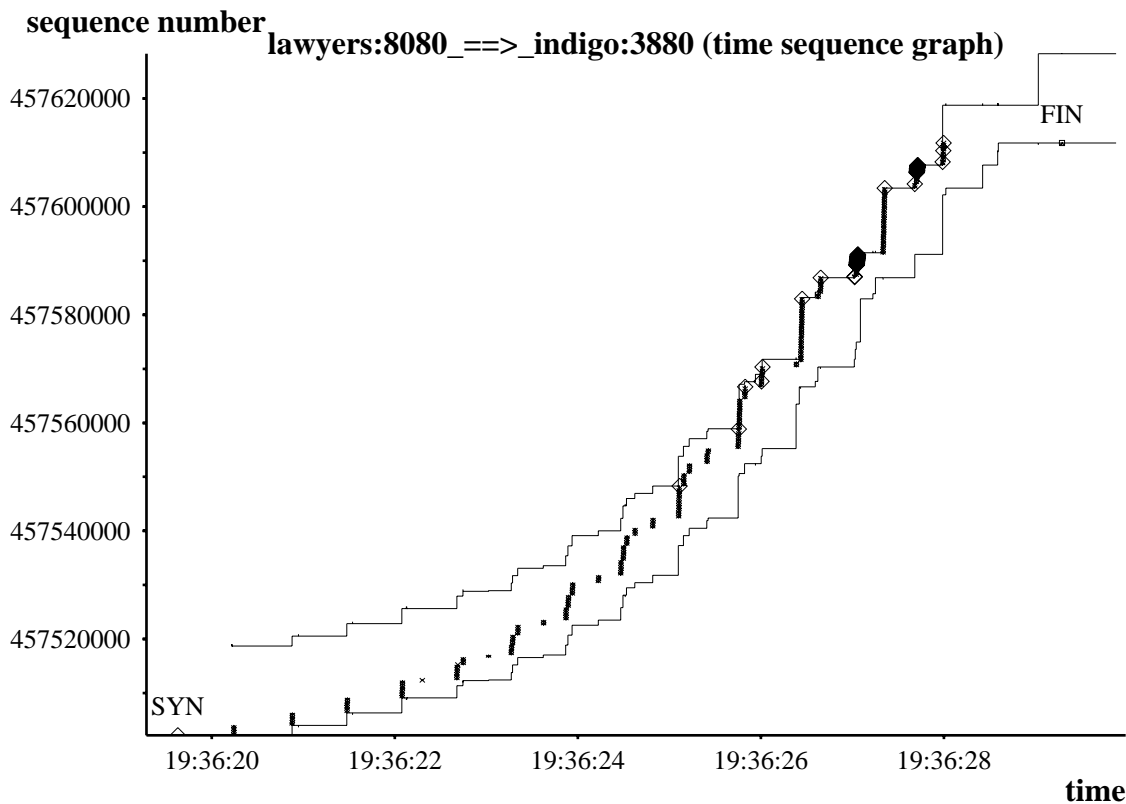


Figure 3.5 HTTP/1.1 Pipeline Time Sequence Graph

Heidenmann's suggestion [NGS⁺97]. This thesis will discuss the influence of the nagle algorithm to the HTTP performance in Section 3.4.

The time sequence graph, Figure 3.5, shows that HTTP/1.1 Pipeline uses the channel capacity more efficiently, but since it disables the nagle algorithm, it produces many unfilled segments.

3.3 Influence of Web Page Constitution

This section compares the performance of HTTP/1.0, HTTP/1.1, and HTTP/1.1 Pipeline for three web pages. In these tests, the efficiency is defined in Equation 3.1, where the *real data bytes* is the total size of the HTML file and all the inline images

for a web page; the *total data bytes sent* is the number of bytes of the network traffic used to get that web page, which includes all protocol headers and retransmitted data packets. The throughput is defined in Equation 3.2.

$$Efficiency = \frac{Real\ data\ bytes}{Total\ data\ bytes\ sent} \quad (3.1)$$

$$Throughput = \frac{Total\ bytes\ sent}{Total\ elapsed\ time} \quad (3.2)$$

From Table 3.3, Table 3.4, Table 3.5 and Table 3.6, we can see the influence of the web page constitution over the HTTP performance:

Table 3.3 Performance Comparisons against CNN Web Page

	packets	bytes	time(secs)	Tput(bps)	eff
HTTP/1.0	480.0	151531.0	15.3	9904.0	0.70
HTTP/1.1	376.8	142713.6	17.5	8173.7	0.74
HTTP/1.1 Pipeline	360.4	141368.8	10.0	14094.6	0.75

Table 3.4 Performance Comparisons against Microsoft Web Page

	packets	bytes	time(secs)	Tput(bps)	eff
HTTP/1.0	423.4	110797.8	15.7	7043.7	0.62
HTTP/1.1	302.2	100384.4	19.6	5134.8	0.68
HTTP/1.1 Pipeline	243.0	95530.0	8.3	11509.6	0.72

Table 3.5 Performance Comparisons against Mini Web Page

	packets	bytes	time(secs)	Tput(bps)	eff
HTTP/1.0	402.8	72574.6	14.8	4903.7	0.40
HTTP/1.1	207.8	56066.4	22.0	2550.8	0.51
HTTP/1.1 Pipeline	115.2	48276.4	7.8	6165.6	0.60

Table 3.6 Performance of HTTP/1.1 Pipeline over HTTP/1.0

	% of packets	% of bytes	% of time	% of Tput	% of eff
CNN	75.08	93.29	65.51	142.30	107.15
MS	57.39	86.22	52.77	163.40	115.88
MINI	28.60	66.52	52.90	125.73	150.25

- The more small size inline images one web page contains, the worse the HTTP performance is. Both the efficiency and the throughput go down if the proportion of small size inline images increases, because the lower level protocol header and TCP setup/close overhead will take higher proportion.
- The more small size inline images one web page contains, the more obvious the benefit of HTTP/1.1 Pipeline over HTTP/1.0, saving more packets, bytes and elapsed time.

3.4 Influence of Nagle Algorithm

HTTP is an application-layer protocol, the communication across the network is controlled by the TCP protocol, hence the nagle algorithm in the TCP protocol plays an important role in the HTTP performance.

3.4.1 When Can TCP Send a Data Segment

At the TCP level, the sender will send a data segment when one of the following four conditions is true [WS95]:

- It can send a full size segment.
- The receiver's window is at least half open.
- No segment that has been sent but not acknowledged, or the nagle algorithm is disabled.
- The sender will close the connection. In this case, the sender will send all the remaining data before closing the connection anyway.

3.4.2 Why Disable the Nagle Algorithm

Heidemann [Hei97] analyses the problem caused by enabling the nagle algorithm (the default) in HTTP/1.1, he calls it *Odd/Short-Final-Segment Problem*. He suggested that the Apache server disable the nagle algorithm and his suggestion was adopted [Hei97].

According to Heidemann, the server writes data at the application-layer in 4 KB chunks. Due to the MSS limit, this chunk of data will be encapsulated into several segments (the default MSS for Ethernet is 1640 bytes and the WAN is 512 bytes or 536 bytes) so that the last one normally will not be a full size segment. In Ethernet, the first two segments will be acknowledged immediately, but the acknowledgment for the third will be delayed for about 200 ms according to the TCP delayed acknowledgment

algorithm. If the server by chance will complete the current connection by sending another small amount of data, the nagle algorithm will forbid it to send because the server has an unacknowledged segment.

In a satellite, as with other long delay communication environments, this problem still exists with some difference: since the RTT is so long, it takes a long time to get the acknowledgment, the last unfilled segment will stay in the output buffer and wait for the acknowledgment of the previous segments, which will take a long RTT about half a second in a satellite environment.

3.4.3 Comparisons: Disable/Enable Nagle Algorithm

HTTP/1.1 suffers a lot if the nagle algorithm is enabled because the client waits for the previous response before sending another request. The last segment of each response is normally unfilled so that the Odd/Short-Final-Segment Problem is met for each request/response, as shown in Figure 3.6.

HTTP/1.1 Pipeline normally will not meet this problem until the response for the final request. In HTTP/1.1 Pipeline, the server normally will get all the requests in the first several RTTs (still in slow-start), so that the server always has data to send, which will help the TCP to accumulate full size segments very quickly (compared to the RTT) until the final segment, as shown in Figure 3.7.

Using the CNN home page as the testing web page, the performance of disabling and enabling the nagle algorithm for HTTP/1.1 Pipeline is shown in Table 3.7. Disabling the nagle algorithm reduces the elapsed time by sending the last segment immediately at the cost of introducing a lot of small size segments as shown in Figure 3.5. It does not use the network efficiently. It is also more likely to cause network congestion.

Enabling the nagle algorithm performs well until the final segment. Delaying the final segment will not only introduce a long delay to the HTTP transaction, but also waste the server's resources: an HTTP server has an upper limit on the number of

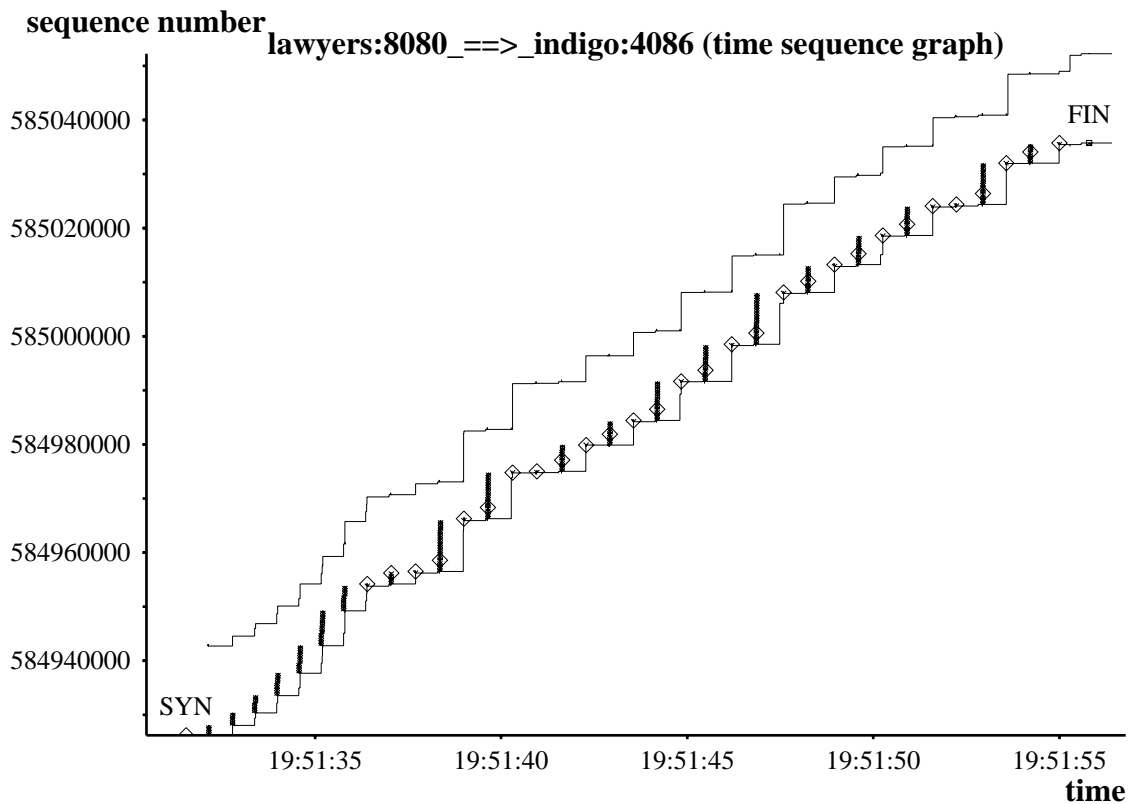


Figure 3.6 HTTP/1.1 Time Sequence Graph (enable nagle)

concurrent connections allowed and each TCP connection will allocate a large memory block.

3.4.4 Suggestions to Solve this Problem

The final segment problem happens only at the final segment for HTTP/1.1 pipeline. The current implementation of HTTP/1.1 Pipeline disables the nagle algorithm for the entire connection. There are several alternative solutions to this problem.

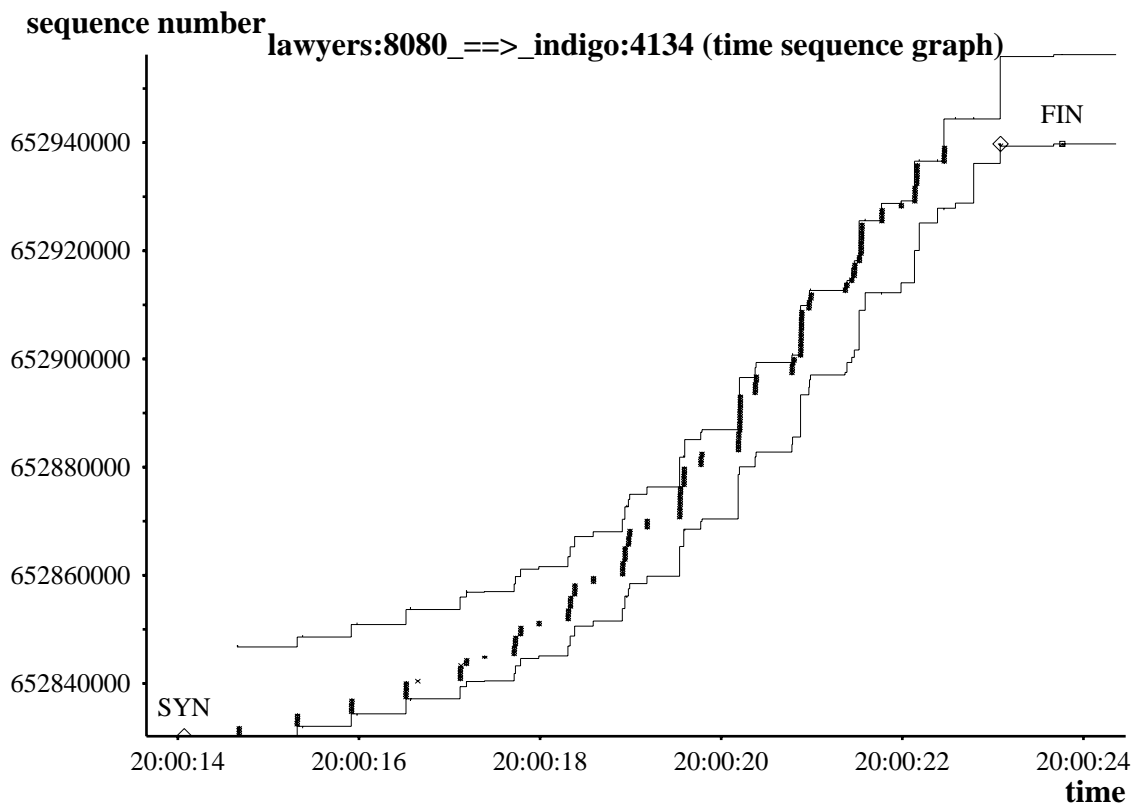


Figure 3.7 HTTP/1.1 Pipeline Time Sequence Graph (enable nagle)

Table 3.7 Performance Comparisons: Disable/Enable Nagle Algorithm

	packets	bytes	time(secs)	Tput(bps)	eff
Totally disable	360.4	141368.8	10.0	14094.6	0.75
Totally enable	312.8	137465.6	10.4	13256.1	0.77
Disable after final response	317.3	138423.6	10.1	13722.3	0.76
Server closes connection	313.4	136824.8	9.64	14193.4	0.77

3.4.4.1 Disable Nagle Algorithm Before/After the Final Response

Disabling the nagle algorithm at any time will force TCP to send all the data in the output buffer as soon as possible (only limited by the congestion window). If we turn on the nagle algorithm (the default) at the beginning of a new connection, and turn it off before/after the server completes the final response, we will use the connection efficiently without any Odd/Short-Final-Segment problem, as illustrated in Figure 3.8.

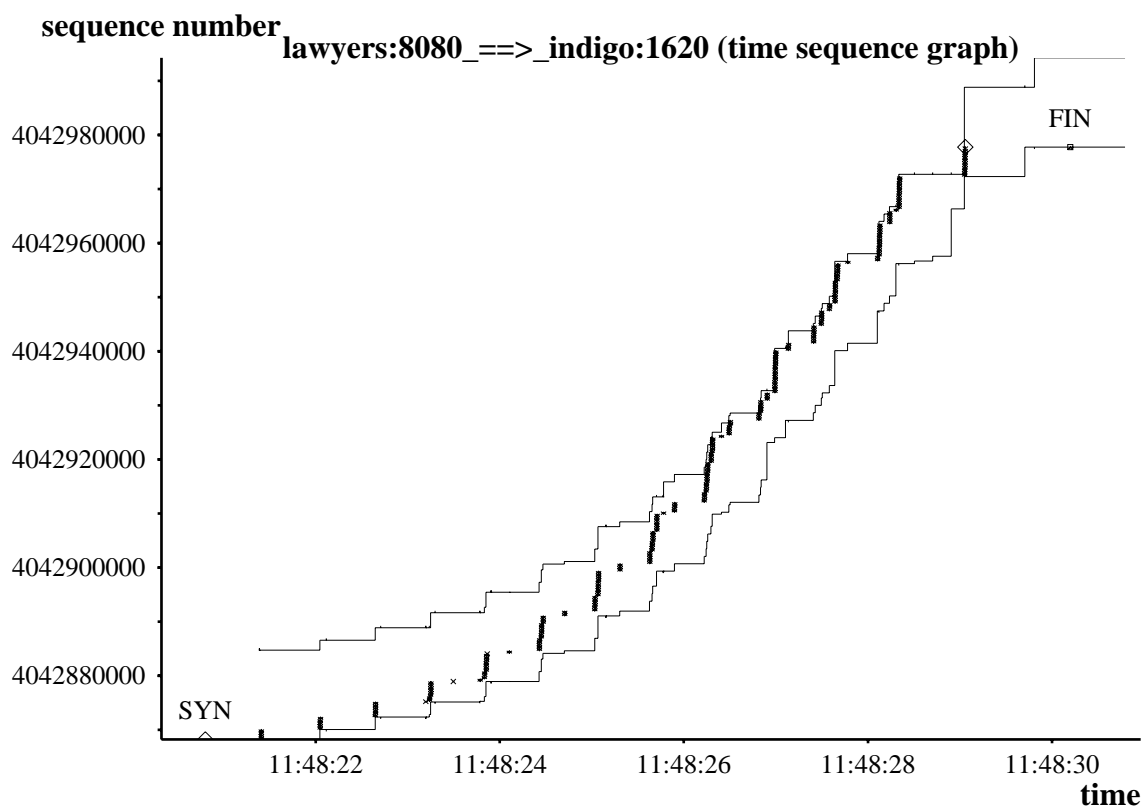


Figure 3.8 Disable Nagle Algorithm after the Final Response

The problem is that the client must tell the server which request is the final request so that the server can know which response is the final response, as discussed

in Section 2.5.2. The client does not know which request is the final request until it receives and parses the whole HTML file because the inline images may appear in any place of the HTML file. If we require that the client indicates which request is the last, the client must wait to send a request until it finds another inline image or to the end of the HTML file. This delay of sending a request may introduce about one RTT delay, just like HTTP/1.1 without Pipeline.

An alternative way is to send any request as before. After the client parses the whole HTML file and sends all the normal requests for inline images, send a separate request to notify the server to turn off the nagle algorithm when it begins to respond to this request (that is, after it completes all the real requests).

3.4.4.2 Server Closes the Connection First

HTTP/1.0 has no Odd/Short-Final Segment problem because there is only one request/response per connection, the server will close the TCP connection after it completes the response. This TCP close action will send all the data in the buffer right away and send a FIN to the peer [WS95].

HTTP/1.1 Pipeline can use the same idea: the client notifies the server which one is the final request by encapsulating a “Connection:Close” HTTP header in the final request. Closing first by the server will not only avoid the final segment problem, but also save about 0.5 RTT, as illustrated in Figure 3.9 and Figure 3.10.

The client also meets the same dilemma: determine which one is the final request. It can do it in the same way as Section 3.4.4.1.

3.4.4.3 Modified Algorithm

So far, the above two solutions are based on the assumption that the HTML file must contain inline images and these inline images do not appear at the bottom of the HTML only. Otherwise, the client will meet the Odd/Short-Final-Segment problem

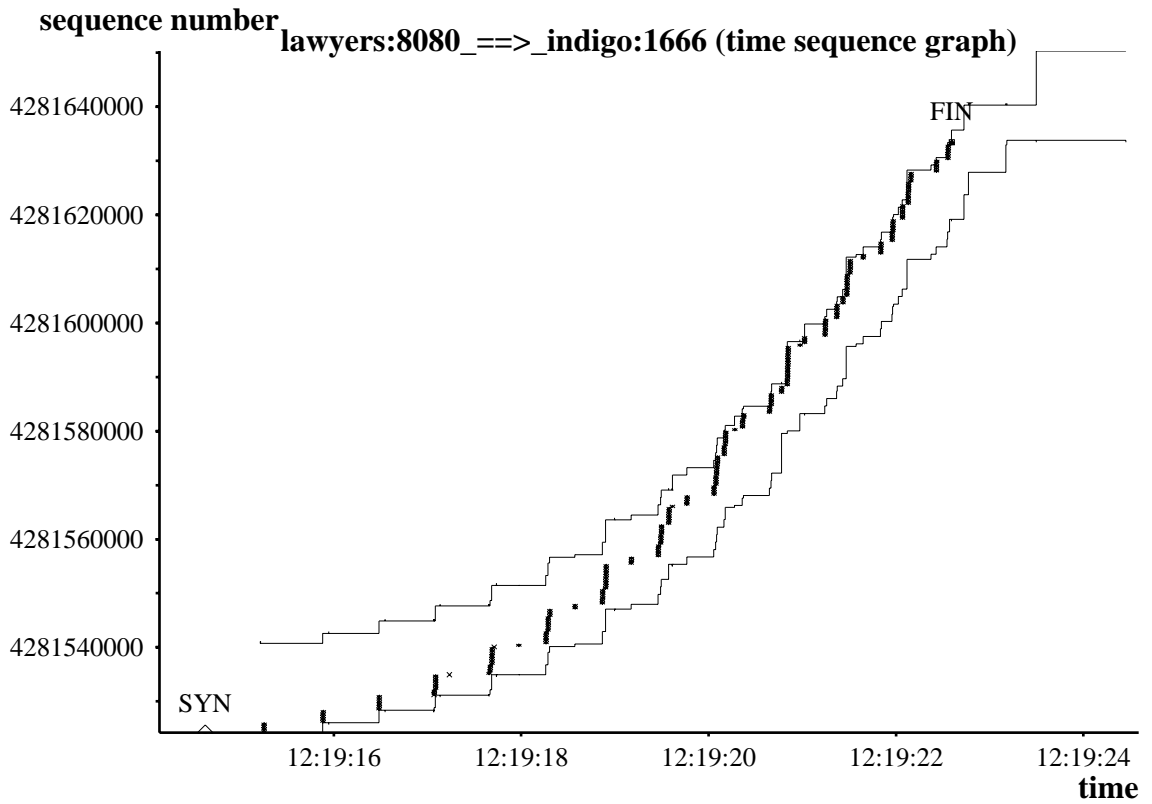


Figure 3.9 Server Closes HTTP/1.1 Pipeline Connection First

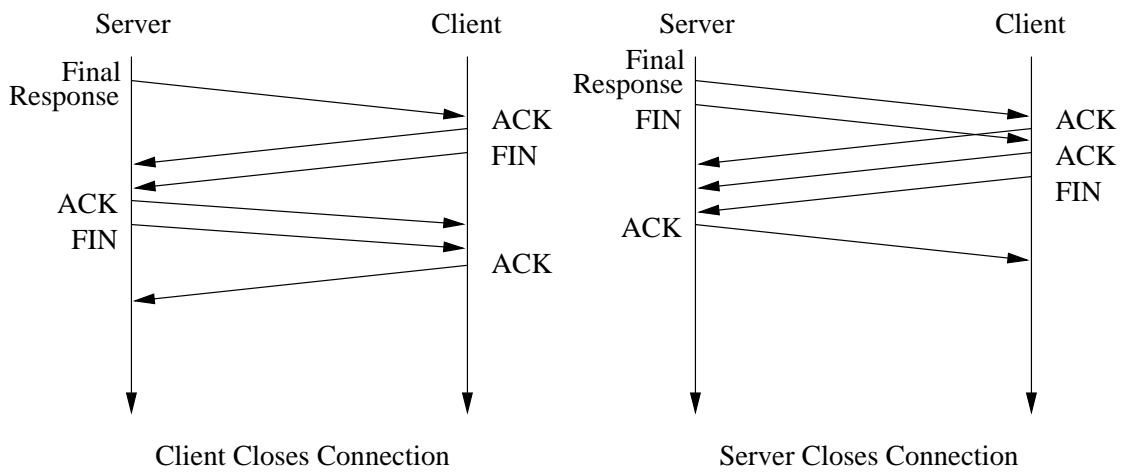


Figure 3.10 Server/Client Closes the Connection

when it receives the HTML file, it will have no chance to tell the server to turn off the nagle algorithm or close the connection.

We can revise the solution as follows.

1. At the beginning of the connection, the server turns off the nagle algorithm. This guarantees that the server can send the HTML file to the client as soon as possible. During this period the server sends only one file (HTML file) and it is in slow-start phase. The throughput is limited by the congestion control so that the server has time to wait to construct full size segments. Disabling the nagle algorithm at this time will not introduce any extra burden to the network.
2. The client parses the HTML file and sends requests as normal. If this is the only URL the client needed, the client closes the connection after it receives the response.
3. When the server begins to handle the first request for an inline image, it turns on the nagle algorithm.
4. The server turns off the nagle algorithm before/after the final request or closes the connection after the final response.

3.4.4.4 Relationship to Current Protocol

The current HTTP/1.1 protocol only defines two kinds of connection control information: `keep alive` and `close`, and this information must be sent along with a URL request. These two kinds of connection control information are not enough for the HTTP/1.1 communication. We should let the client and the server to exchange more information, for example, the information about the nagle algorithm. We should also allow the client and the server to exchange some HTTP control information without requesting a URL.

4. CONCLUSIONS

The one request, one response per connection scheme used in HTTP/1.0 works inefficiently. Each connection suffers from TCP connection overhead and TCP congestion control. HTTP/1.1 improves the network performance by allowing multiple requests/responses per connection. However, HTTP/1.1 spends more time to fetch the same web page than multiple concurrent TCP connections. In order to reduce the elapsed time as well, a pipelining technique must be used in HTTP/1.1.

HTTP/1.1 Pipeline improves the HTTP performance significantly. The effort is particularly obvious if the web page contains many small size inline images. The current implementation of HTTP/1.1 disables the nagle algorithm. It solves the Odd/Short-Final-Segment Problem at the cost of producing many unfilled segment which will lower the network performance.

This thesis puts forward a new solution to that problem: disable the nagle algorithm for the HTML transmission, enable the nagle algorithm for the following inline images, then disable the nagle algorithm after the final response or close the connection by the server. This solution improves HTTP performance further. This suggestion requires HTTP header information which is not defined in the current HTTP/1.1 protocol.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [AK97] M. Allman and H. Kruse, 1997. URL: <http://132.235.67.19>.
- [Apa97] Apache Group. Apache User's Guide, June 1997. URL: <http://www.apache.org/docs>.
- [BLC95] T. Berners-Lee and D. Connolly. Hypertext Markup Language - 2.0, November 1995. RFC 1866.
- [BLFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. Hyper Transfer Protocol – HTTP/1.0, May 1996. RFC 1945.
- [Bra94] R. Braden. T/TCP – TCP Extensions for Transactions Functional Specification, July 1994. RFC 1644.
- [Com95] Douglas E. Comer. Internetworking with TCP/IP, volume I: Principles, Protocols, and Architecture. Prentice Hall, 3rd edition, 1995.
- [Cor97a] Microsoft Corporation, 1997. URL: <http://www.microsoft.com/ie>.
- [Cor97b] Netscape Communications Corporation, 1997. URL: <http://www.netscape.com>.
- [dum94] SunOS 5.5.1 Online Manual, June 1994.
- [FGM⁺97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hyper Transfer Protocol – HTTP/1.1, January 1997. RFC 2068.
- [Gra96] M. Gray. Web Growth Summary, June 1996. URL: <http://www.mit.edu/people/mkgray/net/web-growth-summary.html>.
- [Hei97] J. Heidemann. Performance Interactions Between P-HTTP and TCP Implementations. In *ACM Computer Communication Review*, April 1997.
- [Inc97] Cable News Network Inc., 1997. URL: <http://www.cnn.com>.
- [Mog95] J. C. Mogul. The Case for Persistent-Conneciton HTTP. Technical report, DEC, April 1995. URL: <http://www.research.digital.com/wrl/publications/abstracts/95.4.html>.

- [NGS⁺97] H. F. Nielsen, J. Gettys, A. B. Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley. Network Performance Effects of HTTP/1.1, CSS1, and PNG. Technical report, W3C, June 1997. URL: <http://www.w3.org/TR/NOTE-pipelining-970624>.
- [Nie97] H. F. Nielsen. The Mini Robot. W3C, February 1997. URL: <http://www.w3c.org/Robot>.
- [Ost97] S. Ostermann. Ostermann's tcptrace, September 1997.
- [She97] T. Shepard, 1997. <ftp://mercury.lcs.mit.edu/pub/shep>.
- [Tou97] J. Touch. TCP Control Block Interdependence, April 1997. RFC 2140.
- [WS95] G. R. Wright and W. R. Stevens. TCP/IP Illustrated, volume 2: The Implementation. Addison-Wesley Publishing Company, 1995.

APPENDIX

A. READ HTTP AND TCP GRAPHS

A.1 How to Read an HTTP Transaction Graph

An HTTP transaction graph is a visual representation of HTTP activities. The x-axis is the time index. The y-axis is the URL index, ordered by the sequence that URLs appear.

The lowest line shows the SYNs and FINs for a TCP connection. The HTTP transaction based on this TCP connection is shown above this line. Each line represents the activities for one URL. The first diamond indicates the time that the request for that URL is seen, the text left to that diamond indicates the the name of that URL and HTTP protocol version. The horizontal line to the right of that diamond stands for the response to that request (from start to finish). The second diamond indicates the time that the acknowledgment for the last segment of that response is seen.

A.2 How to Read a TCP Time Sequence Graph

TCP time sequence graph is a graphical representation of a TCP connection. TCP connections are full duplex, a TCP time sequence graph shows the activities in one direction. A TCP connection can be represented by two related TCP time sequence graphs, each of which represents one direction.

The x-axis is the time index of a TCP connection. The y-axis is the sequence number. A data segment is a vertical line with arrows at both ends. The length of the vertical line stands for the length of that segment. A segment with a diamond on the top means this segment is sent by a TCP PUSH operation.

Below the data segments, there are horizontal lines. They stand for the base of sender's output sliding window. The vertical line between two horizontal lines means that the sender advances its sliding window.

Above the data segments are horizontal lines. They stand for the upper limit of the receiver's window. The vertical line between two horizontal lines means that the server advances its sliding window.

The space between the upper and lower horizontal lines stands for the amount of data that the receiver allows at that time.

B. WEB PAGES

This appendix shows the contents of Mini Home Page, CNN Home Page and Microsoft Home Page used in this thesis.

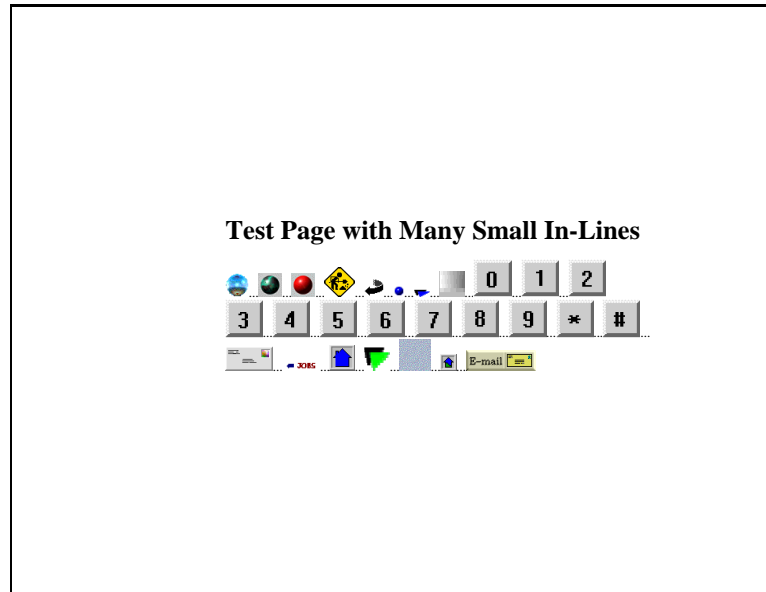


Figure B.1 Mini Home Page



Figure B.2 CNN Home Page (part 1)



Figure B.3 CNN Home Page (part 2)

attack risk

SPECIALS

- The OKC Trials: The U.S. vs. Terry Nichols
- Viva Elvis!
- Destination Mars
- India-Pakistan: 50 Years of Independence

VIEWS

- Bill Mitchell's Cartoon: "Souls are getting cheaper everyday," quipped Satan.
- AllPolitics: Pundits and Prose

CNN PLUS

- Discuss college drinking
- Labor Day meal ideas
- Subscribe to Plus Mail

STYLE

- Unusual materials make up Fendi's accessory line
- Fabrics recall romance of high seas travel

COMMUNITY

- The Board: What are your thoughts on air bags in motor vehicles?
- Chat with us! 9 AM to 11 PM EDT

TRAVEL

- International tourists discovering Florida Panhandle
- Make your travel plans online
- Is Clintons' Vineyard vacation a dud?
- A bird-brain idea for getting photos back fast
- Destination: When in Rome, indulge in ice cream
- Lead us to your best biking trails

EARTH

- Amazon Indians meet to protest rain forest destruction

FRINGE

- He thought they said hide and sleep...
- Anarchist festival flares in Nevada desert
- Japan goes ape for waterskiing monkeys

CUSTOM NEWS

- AOL urges its members to protect themselves
- S. Korea develops world's first anti-hacking program
- Marimba and Netscape push new WWW standard
- Still want more Tech news?

Figure B.4 CNN Home Page (part 3)

Smash it. IBM

FEEDBACK


- Email us your comments
- What you said!

PAGER

- Get automatic updates

AIRMEDIA

- Download a free demo



JOBS


- CNN Interactive Interns
- CNN/SI Producer
- CNN/SI Copy editor
- CNN/SI Multimedia designer
- CNN/SI ENG/SNG Engineer
- CNN/SI Software Developers

CUSTOM NEWS

- Register for your personal news

POINTCAST

- Download it free!



Back to the top
 (C) 1997 Cable News Network, Inc.
 All Rights Reserved.
 Terms under which this service is provided to you.

Figure B.5 CNN Home Page (part 4)

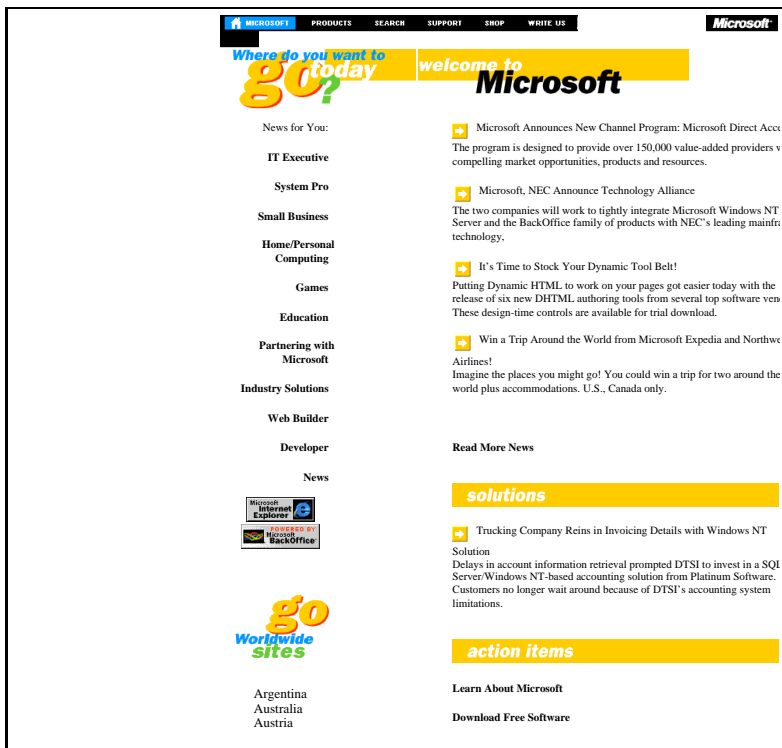


Figure B.6 Microsoft Home Page (part 1)

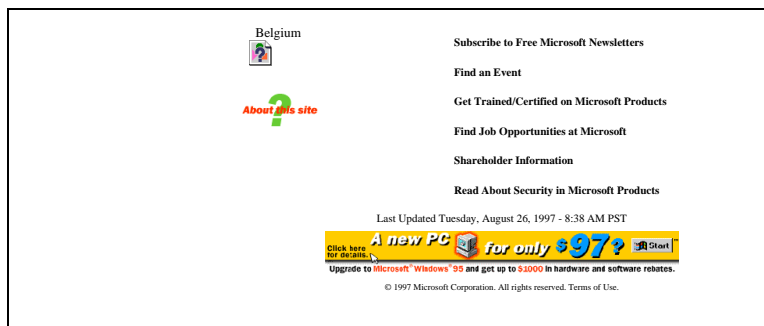


Figure B.7 Microsoft Home Page (part 2)

Chen, Xin. M.S. November, 1997
Electrical Engineering

Performance Analysis of Persistent HTTP over Satellite Links (32 pp.)

Director of Thesis: Dr. Shawn Ostermann

HTTP/1.0 uses one request/response per TCP connection. Due to the TCP connection overhead and the TCP congestion control, this scheme works inefficiently. Persistent HTTP (HTTP/1.1) improves the network performance by allowing multiple requests/responses per TCP connection. In order to reduce the total elapsed time, pipelining technique must be used in HTTP/1.1. This thesis investigated the performance of HTTP/1.0, HTTP/1.1 and HTTP/1.1 Pipeline in satellite links by analyzing the TCP traffic data between the client and the server. HTTP/1.1 Pipeline improves the HTTP performance significantly. The effort is particularly obvious if the web page contains many small size inline images. Current implementations of HTTP/1.1 disable the nagle algorithm, solving the Odd/Short-Final-Segment Problem at the cost of producing many unfilled segment which will lower the network performance. This thesis suggests that the client and server exchange more information so that the server can disable the nagle algorithm or close the connection after the final response, this will improve the HTTP performance further.

Approved: _____