Appeared in Proceedings of IEEE Globecom '98 Internet Mini-Conference, Sydney, Australia, November 1998.

# BENEFITS OF TRANSPARENT CONTENT NEGOTIATION IN HTTP

**Srinivasan Seshan**
*srini@watson.ibm.com*

*IBM T.J. Watson Research Center*
*Yorktown Heights, NY 10598*

**Mark Stemm, Randy H. Katz**
*{stemm,randy}@cs.berkeley.edu*

*Computer Science Division*
*University of California at Berkeley*
*Berkeley, CA 94720*

## Abstract

Over the past several years, the popularity of the World Wide Web (WWW) has steadily increased. Unfortunately, as a result of overloaded servers and lack of bandwidth between clients and servers, response times observed by end users have also steadily increased. Response times are often so unbearable that the popular press has relabeled the "World Wide Web" the "World Wide Wait". In this paper, we propose the use of *content negotiation* to allow clients and servers to tune response times to meet user demands. In our system, a client can negotiate content to match the bandwidth available between the server and itself. Clients obtain information about the available bandwidth from a SPAND performance prediction server [10]. When the available bandwidth is low or the original page is large, the client requests a smaller version of the document to reduce the response time. Our experiments show that the client is able to identify the subset of transfers that will perform poorly and that transferring a smaller amount of information on these connections helps improve the overall response time. In our system, servers that are limited by the bandwidth leaving their site reduce the quality and thereby the size of the documents being transferred. This allows the servers to serve many more requests when they are under heavy load. Our measurements show that the benefits can be significant, but are very dependent on the ratio of negotiable to non-negotiable content. For example, in a typical mix of 60% image or other negotiable content, a server can increase the number of requests it handles by 50%. When 90% of the content is negotiable, a fourfold increase in Web server throughput is possible.

## 1. Introduction

The Hypertext Transport Protocol (HTTP) is the most commonly used application-to-application transport protocol for World Wide Web (WWW) applications. The response times that clients experience while using HTTP is related to the available bandwidth along the network path from a client to a particular server, the number of other clients that are also contacting the same server, and the size of the pages being transferred. Designers of a Web site often make assumptions about the number and type of clients that will visit them and design their site appropriately. For example, if the typical client is expected to have high available bandwidth, the site may consist of large, full color images or streaming multimedia. If the typical client is expected to have low available bandwidth, then the site may consist of smaller low color depth images without streaming multimedia. In addition, Web site designers allocate bandwidth to the Internet in proportion to the number of clients that are expected to visit the site.

Unfortunately, these assumptions about a client's available bandwidth and the total client request load are often wrong. If the available bandwidth to a client is lower than the estimate determined by the Web site creators, then the client will have to wait an excessive amount of time to download a particular Web object. Unfortunately, due to the heterogeneity of today's Internet, there is no such thing as a "typical" client, and some clients visiting a Web site will have longer download times than those predicted by the site designers. In addition, the Internet connection of many Web servers limits the available bandwidth to all clients when they are swamped with an unexpected burst of requests from a large number of clients. The response times of clients increases as the connection from the server to the Internet becomes a bottleneck. *WWW clients and servers need a mechanism to adapt WWW content to match dynamic changes in available bandwidth.*

To overcome this problem of bandwidth heterogeneity, the IETF HTTP Working Group and the World Wide Web consortium has defined mechanisms in HTTP for *Transparent Content Negotiation*. This allows a client and server to negotiate features of a Web object including the content fidelity. For example, clients with higher available bandwidth can request large full-color versions of images, and clients with lower available bandwidth can request smaller black and white versions of images. The idea of changing object fidelity to match network characteristics is not unique to HTTP. Related work has experimented with different ways to achieve the same goal by maintaining or generating multiple representations of Web content [9] [5] [4], sometimes in ad-hoc ways. For example, the administrators of the CNN Web site www.cnn.com turn off advertisements during periods of heavy client traffic as an attempt to reduce server load.

Using HTTP content negotiation, clients and servers can modify content fidelity to obtain acceptable response times. Clients may initiate a change in content fidelity to obtain a fixed response time at the cost of content quality. Servers may initiate change in content fidelity to reduce the total bandwidth leaving a server complex as the number of client requests per second increases.

In this paper, we examine the effectiveness of HTTP content negotiation at reducing the actual response times of WWW clients and handling variable request loads on WWW servers. We answer the following specific questions:

1. How much response time can be reduced by changing content fidelity? Are transfer sizes too small for content negotiation to help?
2. Client-side initiated negotiation requires some estimate of available bandwidth and other network characteristics to work. How often is this estimate accurate, and do clients obtain the desired fixed response times?
3. For server-side negotiation, can outgoing bandwidth stay relatively constant as client load is added? What is the increase in Web server throughput by using content negotiation?
4. What is the overhead of using content negotiation in a real Web server implementation?

To examine the effectiveness of client-initiated content negotiation, we perform trace analysis of actual client traffic. To examine

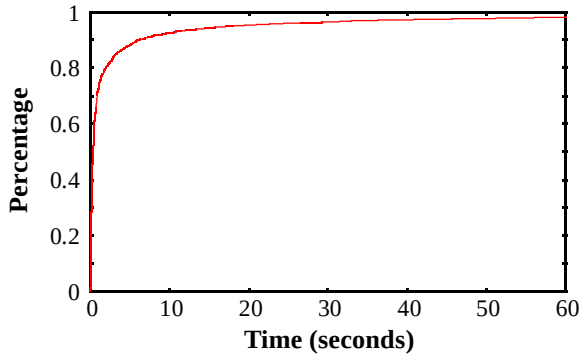


**Figure 1. CDF of transmission times for pages retrieved by clients at IBM research from servers in the Internet.**

the effectiveness of server-initiated content negotiation, we use an implementation of IETF Content Negotiation in Apache, a commonly used Web server, and stress the implementation by generating requests from Surge, a Web client request generator.

The rest of this paper is organized as follows. In Section 2, we discuss the problems of long client-side response times and server-side bandwidth shortages. Section 3 provides details about the IETF Content Negotiation protocol and Apache's implementation of it. Section 4 analyzes the effectiveness of client-initiated content negotiation at reducing Web page download times. Section 5 describes the benefits of server-initiated content negotiation in handling additional request load. Finally, in Section 6, we present conclusions and future work.

## 2. Motivation

In this section, we examine the problem of long variable response times for clients and bandwidth bottlenecks on servers. We do this by measuring typical client response times and characterizing the behavior of servers under heavy load.

To understand the response times observed by clients today, we recorded a number of Web page transfers by clients at IBM research and examined the variation in available bandwidth to distant sites. The IBM research clients consisted of mainly workstations and PCs connected via ethernet or token ring to the internal network. All clients were at most a few network hops from IBM's connection to its Internet service provider.

The response time for a page was measured by determining the difference in time from when the Web page was requested to when the last Web object embedded in the page was transferred. Figure 1 plots the CDF of response times for these page transfers. Over 10% of the transfers completed in under 0.1 sec. and over 10% took over 6 sec. to finish. We also measured the bandwidth of each connection used to transfer these Web objects. Figure 2 shows the CDF of the average bandwidth of these transfers. About 10% of the transfers had a bandwidth of less than 6 Kbit/sec and 10% had a bandwidth of greater than 220 Kbit/sec These measurements show us that clients do experience a wide variation in performance and response times. Since the IBM community was very homogenous and well connected, this wide variation can be attributed to the performance of the server and the available bandwidth along the network path to the server.

To show that the Internet link to a well-designed server is likely to be the bottleneck of the system, we examined the published SpecWeb96 benchmark results (available at http://www.spec.org/osg/web96/) for Web servers offered by a variety of companies. For the 5 fastest single-processor configurations reported as of

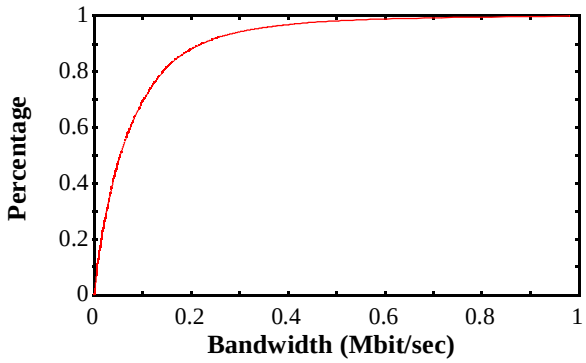


**Figure 2. CDF of bandwidth for transfers between clients at IBM Research and servers in the Internet.**

| Web Server | SpecWeb Throughput (ops/sec) | Estimated Aggregate Bandwidth (Mbytes/sec) |
|---|---|---|
| IBM Netfinity 5500 running Apache 1.3.1 | 2575 | 41.5 |
| NetServer LH 3/400 running Netscape 3.5 for Netware | 2131 | 34.3 |
| Digital Alphaserver 1200 5/533 running Zeus 1.3 | 2045 | 32.9 |
| Compaq Proliant 3000 running Netscape 3.5 for Netware | 1639 | 26.4 |
| Digital Alphaserver 1200 5/533 running Zeus 1.2.7 | 1631 | 26.3 |

**Figure 3. Bandwidth transmitted by different servers under the SpecWeb benchmark**

August 13, 1998, we multiplied the benchmark throughput of operations/second by the average size of a document in the benchmark (approximately 16 kilobytes), resulting in an average bandwidth requirement for that Web server. The results of this calculation are shown in Figure 3. We see that a single-processor Web server can almost saturate the capacity of a 45 Mbit/sec T3 line. As most Web sites are connected through smaller than T3 links, this indicates that a single Web server can easily handle a large number of clients accessing it and the connection to the Internet is likely to become the first bottleneck of the system. For sites that are connected through multiple T3 links, distributing the load across multiple servers or using multiprocessor configurations would easily allow the Web Server site to match the available bandwidth of its Internet connection.

From these measurements we see that bandwidth at the server and in the network is the likely cause of variations in response times observed by clients. This indicates that by controlling the size of transferred pages, and as a result the quality, we can control the response times observed by the clients and the load on the server's network connections.

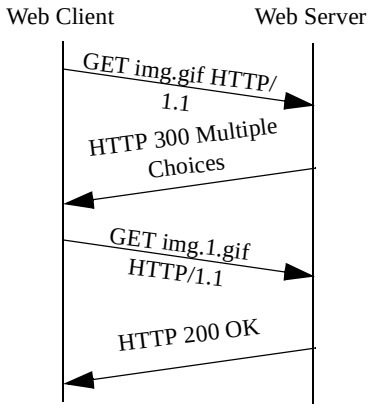


**Figure 4. Sample Transaction using Transparent Content Negotiation**

```
HTTP 300 Multiple Choices:
Date: Tue, 11 Jun 1996 20:02:21 GMT
    TCN: list
    Alternates:
{“img.1.gif” 1.0 {type image/gif} {length 4000}},
{“img.2.gif” 0.75 {type text/html} {length 3000}},
{“img.3.gif” 0.5 {type image/gif} {length 2000}}
```

**Figure 5. Sample HTTP Response for Transparent Content Negotiation**

## 3. HTTP Transparent Content Negotiation

This section describes the details of the IETF Transparent Content Negotiation Protocol and Apache’s implementation of the protocol. A complete description is provided in [8].

### 3.1 How Transparent Content Negotiation Works

Sometimes Web objects are available in alternate representations. For example, a text file may be available in several languages, an image may be available in several sizes, or a paper may be available as a postscript document or an HTML page. Transparent Content Negotiation is a mechanism that allows a client and server to select the most appropriate variant for a particular client. The word “Transparent” is used because both the client and server are aware of the available alternate representations. HTTP also supports Non-transparent Content Negotiation, where only the server is aware of the available representations. Typically, clients and servers can negotiate on the following dimensions of an object:

- Mime Type
- Language
- Character Set
- Encoding
- Length

Negotiation can also be done on other arbitrarily defined feature tags (for example, screen size, color depth, etc.). Separate work is being done on a “Feature Tag Registry” [7] that allows developers to define Feature Tags in a unified framework similar to that used for MIME types [3].

Figure 4 shows a typical transaction using Transparent Content Negotiation. A Web server responds to a request for a Web object with a “HTTP 300 Multiple Choices” response that lists the variants and their features. Figure 5 shows the format of a Multiple Choices response. The Web client uses this response to select one of the variants and downloads that variant from the Web server. There are also mechanisms that allow a client to execute a predefined variant selection algorithm on the Web server (for example, to automatically choose the smallest acceptable object), avoiding the extra round trip associated with the Multiple Choices response.

In this paper, we only consider negotiating the size of image objects[1]. This allows a client to trade off image size, quality, and color depth for downloading speed and a server to trade off image quality for greater throughput under periods of high load.

### 3.2 Content Negotiation in Apache

In this section, we describe how content negotiation is implemented in Apache and how we modified it for our purposes. The version of Apache we used only implements Non-transparent Content Negotiation. For our purposes, this was acceptable, because we used the Apache implementation to measure server-side negotiation where the client does not take part in the variant selection process.

Apache has two mechanisms to enable negotiation of Web content: *Type Maps* and *MultiViews Searches*. A Type Map is a file that explicitly lists the variants to be negotiated and their characteristics (e.g. size, quality, language, mime type). When a Web client requests a Type Map, the Web server uses the Type Map to find the characteristics of the variants and selects the variant that is most appropriate for that particular client. Using MultiViews, a Web server does an implicit filename pattern search and chooses among the results. For example, if a Web client requests a file “img.gif” and no such file exists on the server, the server looks for files with names “img.*” and internally constructs a Type Map using those files. If the Web server happens to find a Type Map file while doing this wildcard search, then that Type Map is used instead. Figure 6 shows examples of negotiation using Type Maps and MultiViews Searches

We used Apache’s MultiViews mechanism for content negotiation. We chose to use this rather than Type Maps because it allows us to avoid changing HTML documents. For example, if we used Type Maps, we would have to modify HTML documents to change all hyperlinks to images and embedded image references to point to the Type Map instead of the original images. To make a object negotiable, we changed the name of the stored object to a different one. When a client requested that negotiable object, it forced a MultiViews search. To speed up the MultiViews process, we modified the Web server to explicitly look for a Type Map file before doing the filename pattern search. This avoided a linear search through the files in a directory.

### 3.3 Web Content Crawler

An important part of a content negotiation implementation is a program that generates the alternate representations among which the client and server choose. To address this, we wrote a Web content crawler that scans the files on a Web server and generates alternate representations. As previously mentioned, we only consider negotiation of image files. To generate alternate representations for gif and jpeg image files, we used transcoding programs written by the GloMop group at UC Berkeley for their TranSend WWW proxy [5].

---

1. For the measurements in Section 5, we actually negotiate the size of text objects, because the request generation program we used generates and requests text files.

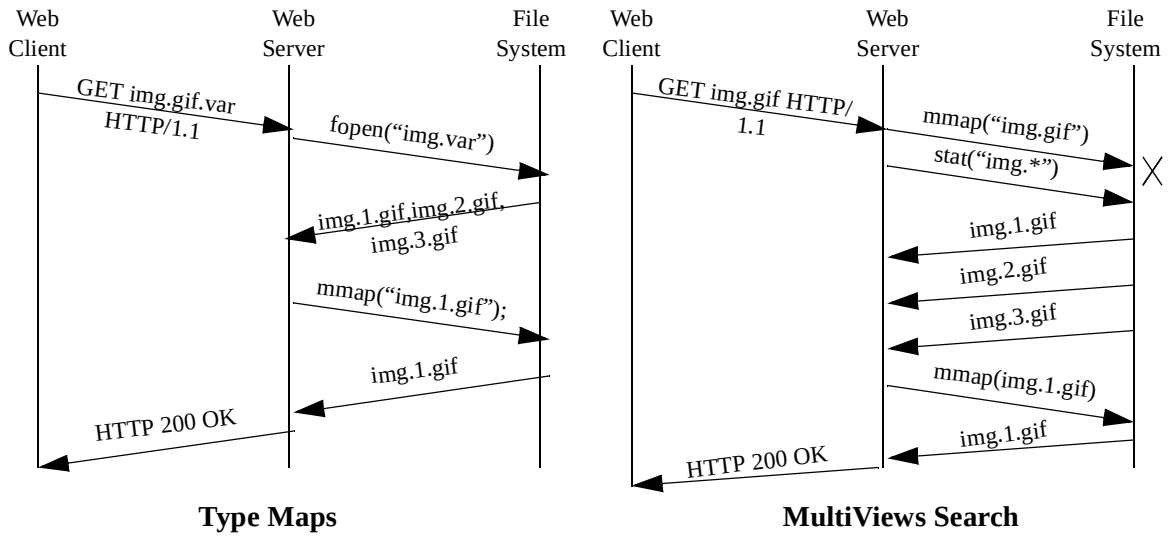


**Figure 6. Apache Mechanisms for Retrieving Negotiated Documents**

Our Web content crawler generated 6 alternate representations for each image file, each of a different size. Each alternate was 70% of the size of the next larger alternate, leading to possible alternates that are 70%, 49%, 34.3%, 24%, 16.8%, and 11.8% of the size of the original image. We chose a multiplicative decrease in size rather than a linear decrease in size to keep the relative size between representations the same. For example, if we had chosen representations that were 20%, 40%, 60%, and 80% of the size of the original image, the relative size of the first alternate to the full size image is 80%, whereas the relative size of the smallest two alternates is 50%.

## 4. Client-initiated content negotiation

As shown in Section 2, the transfer time for retrieving Web pages from different servers varies greatly. This is undesirable since users would like consistent response times for viewing similar information. This variation is primarily a result of differences in available bandwidth to the server site, speed of the server and size of the requested pages. Since there is no easy way to modify available bandwidth or the speed of the server, we must resort to modifying the size of Web pages to provide unchanging response times. In this section, we explore the effectiveness of our algorithms to provide this consistent performance.

### 4.1 Algorithm

The basic technique for providing constant response times consists of the following steps. The client obtains a list of alternate versions of the page. The client then retrieves a performance estimate for the available bandwidth to the server. Based on this performance prediction and the size of the variants, the client estimates the transfer time for the different versions of the Web page and chooses the one that most closely matches the user's requested response time.

For the client to estimate the transfer time for a Web page, it needs to know the combined size of the base HTML page and its embedded objects. We propose to add a feature tag called *full-page length* to the base HTML page. Alternate versions of the HTML document have different quality and different total page sizes. The client obtains a full list of the variants and total sizes via the Transparent Content Negotiation mechanism.

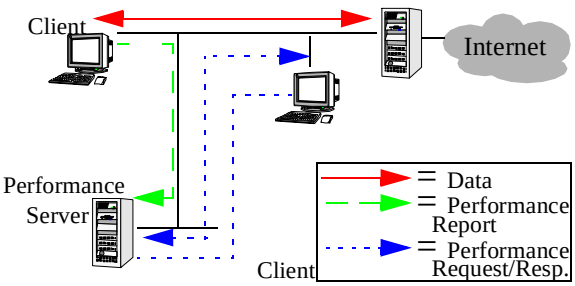


**Figure 7. Design of SPAND**

#### 4.1.1 Using SPAND to obtain Performance Estimates

In addition to the list of alternate views of the page, the client needs an estimate of the performance to the server to determine which version to transfer. We chose to use the SPAND performance prediction system [10] to provide these estimates. SPAND (**S**hared **PA**ssive **N**etwork performance **D**iscovery) is a network performance prediction system that allows a group of similarly connected clients to pool information about the network characteristics of the path to a distant network host. As clients communicate with distant hosts, they send *Performance Reports* that summarize their connectivity to that host to a *Performance Server*. Later, clients send a *Performance Query* to the Performance Server asking about the connectivity to a distant host. Using the collection of Performance Reports, the Performance Server responds with a *Performance Summary* that indicates the typical performance to the distant host. Figure 7 shows a typical SPAND transaction.

### 4.2 Measurements

In order to determine the effectiveness of the client content negotiation algorithms, we chose to simulate their effect on a recorded trace of Web accesses. This simulation required a number of steps. A packet trace had to be collected and converted into a series of Web page transfers. The performance for these transfers was predicted and, if necessary, the page requests were converted into requests for alternate versions of the page. We then determined the actual time that would have been necessary to download the alternate version of the Web page.

A machine at the connection between IBM Research and its Internet service provider collected the analyzed trace. This machine used tcpdump to record all packets sent from or to the HTTP port (port 80). The trace was collected over a 4 hour period on a Sunday. It contained a total of 8618 Web page retrievals and 23719 Web object transfers. We modified tcpdump to search each packet sent to a Web server for HTTP headers and record any GET requests. It also searched packets from Web servers containing HTML content for any embedded objects. Since each packet was independently parsed, (i.e. the entire Web transfer/object was not reconstructed) referenced files or requests that were split across multiple packets could not be parsed by the modified tcpdump program. Later steps used heuristics to reconstruct this missing information.

The next stage of the process matched up embedded HTML objects and GET requests to recreate the transfers of entire Web pages. When a Web object was requested, its transfer was associated with the earliest retrieved HTML page that embedded it. If the transfer of an embedded object was never observed, we assumed that the client used a locally cached copy of the object. We used a simple heuristic to associate any remaining unmatched Web objects transfers to a Web page. Each unmatched object was assumed to belong to the most recently requested Web page transferred between the same client and server. This correctly repaired the majority of references that were split amongst multiple packets.

The predicted performance for each Web page transfer was obtained by playing back the recorded trace file to the SPAND Performance Server. After each connection, a report about its performance was sent to the SPAND server. Before each page was requested from the server, the SPAND server was queried for the performance between the server and client. We used the following heuristic formula, which combines the size of the Web page transfer with the performance predictions, to estimate the duration of the transfer:

$$\left(\frac{3}{2} + log_2\left(\left\lceil \frac{size}{mss} \right\rceil\right) + 1\right) rtt + \frac{size}{bw}$$

*Size* is the total size of the page, *rtt* is the round trip estimate to the server and *bw* is the estimated bandwidth between the server and client. The first term in the formula multiplies the estimated number of round trips for a TCP connection of *size* bytes, $(3/2 + log_2(\lceil size/mss \rceil) + 1)$, by the *rtt* estimate. This estimate of round trips is composed of the overhead for the initial 3-way handshake (3/2 round trips) and the number of rounds of slow start necessary to transfer $\lceil size/mss \rceil$ packets during slow start$(log_2(\lceil size/mss \rceil) + 1)$. This part of the formula assumes that each packet is acknowledged (i.e. no delayed acknowledgments), no losses occur, the remote server starts with a congestion window of one packet and the transfer never leaves the slow start phase. The second term divides the size by the available bandwidth between the sites. We assume that the server made available multiple representations of the same page that were 1/2, 1/4, 1/8, 1/16 and 1/32 of the original page's size. If the transfer duration estimate exceeded the target retrieval time (in our experiments 10 sec.), we chose the largest possible representation that we estimated could be transferred in the target time.

Once a representation of a page was chosen, we examined the traces to determine how long it would have actually taken to transfer that representation of the page. The size of each embedded object was adjusted by the appropriate factor. The tcpdump trace was examined to determine how long it took for each objects' transfers to transmit at least the necessary number of bytes. We used the sum of these transfer times to estimate the total time needed.

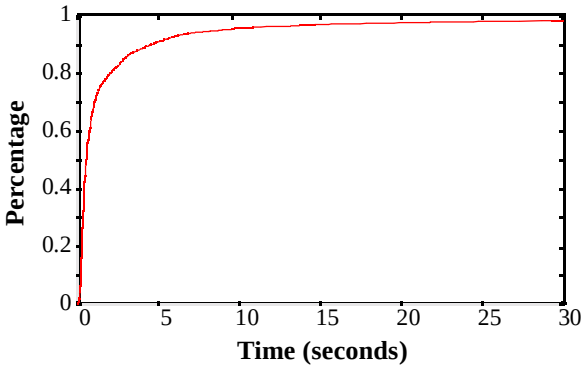


**Figure 8. CDF of actual transmission times for pages that had performance estimates but did not require retrieval of an alternate version.**

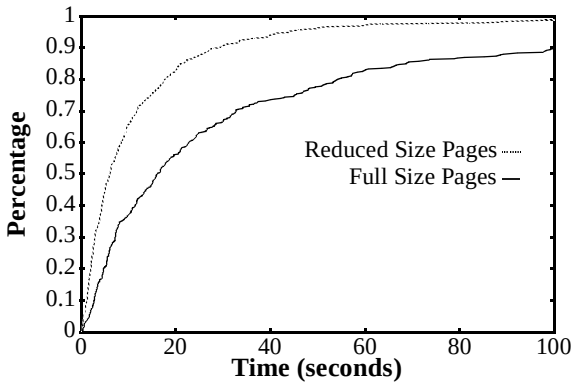


**Figure 9. CDF of transmission times for pages that had performance estimates and required retrieval of an alternate version. Retrieval times for the original page and the negotiated version requested are shown.**

### 4.3 Results

The collected trace contained a total of 8618 page transfers. Of these, performance estimates were available for 3854. If the trace had been collected for a longer period, a larger percentage of the transfers would have had estimates. Of these transfers, the predictions for 3424 of the transfers indicated that their transfers would complete in less than the desired retrieval time. The actual time to transmit these pages is shown in Figure 8. Over 95% of these requests completed in well under the target time of 10 sec, meaning that our system determined correctly that no negotiation was necessary. The remaining 430 transfers were expected to take more than 10 seconds and smaller versions of the pages were requested. Figure 9 shows the actual completion times for the page retrievals under two scenarios: (1) where content negotiation was used to retrieve smaller versions of the pages, and (2) where no content negotiation was used and the full-sized version of the page was retrieved. The median time taken for the full page transfers was about 16 sec, whereas the median negotiated page transfer only took 6 sec. We can see that negotiation reduces the download time. It also does a fairly good job at keeping the actual download time at or below the target time. Using content negotiation, 60% of the downloads completed in under 10 seconds and over 90% of the requested pages arrived in under 29 sec. A significant number of

the non-negotiated page transfers took much longer with 10% taking longer than 101 sec.

These results show that the client negotiation algorithms properly select the subset of page transfers that perform poorly. In addition, transferring smaller versions of these pages makes them meet the user's response time desires. Although approximately 60% of Web pages arrive in under 10 seconds (the target time), using our approach does not always guarantee that Web pages will arrive in the target time. This is due to the inherent variation in characteristics such as round trip time and available bandwidth that change from minute to minute. Although SPAND does a fairly good job at predicting performance, the Performance Responses given by SPAND still have error associated with them, leading to errors in the estimated size reduction required to download a page within the target time.

## 5. Server-initiated content negotiation

In this section, we quantify the benefits of server-initiated content negotiation in managing bandwidth requirements of busy Web servers. This allows a Web server to handle additional load in cases where the connection between the Web server and the Internet is the bottleneck link.

### 5.1 Implementation

Our implementation consists of two parts: adding mechanisms to the Web server to change the size and quality of Web objects served (and therefore the bandwidth leaving the server), and a separate policy program that watches and manages the bandwidth leaving the server.

To provide a mechanism for changing the quality of Web objects, we added a configurable *Maximum Size* option to the variant selection algorithm. The Maximum Size is a fraction between 0 and 1 which indicates the largest allowable variant that is selected by Apache. For example, if the Maximum Size fraction is 0.5, only variants that are less than one-half the size of the original object are considered when selecting a variant. By modifying the Maximum Size fraction, we can proactively control the size of the objects being delivered by the Web server, and therefore the total bandwidth leaving the server machine.

The policy program is a separate program that watches the total bandwidth leaving the Web server and reduces the quality of objects served by the Web server if the total bandwidth becomes too great. We modified the device driver for the network interface of the server to report the total number of bytes transmitted over the interface. The policy program read this statistic every fifteen seconds to obtain an estimate of the bandwidth of the outgoing link from the Web server. This statistic was exponentially smoothed resulting in an average bandwidth *sbw*. The goal of the policy program was to keep *sbw* between a high water mark that was 90% of the limit bandwidth and a low water mark that was 50% of the limit bandwidth. If *sbw* rose above the high water mark, the Web server was restarted by the policy program with a lower Maximum Size fraction. If *sbw* fell below the high water mark, the Web server was restarted with a higher Maximum Size fraction. Whenever the Web server was restarted, a minimum of two minutes passed before changing the Maximum Size fraction again, to allow enough time for the changes to take effect.

### 5.2 Experimental Setup

Our experimental setup consisted of a Web client machine connected to a Web server machine through a router (Figure 10). The Web client and server machines were 200 Mhz Pentium Pro PCs running Linux 2.0.30, and the router was a 133Mhz PC running

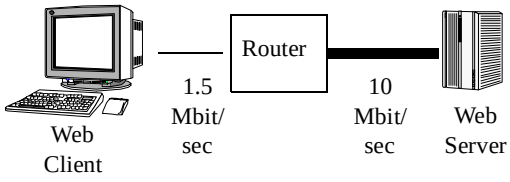


**Figure 10. Topology for Server-side Experiments.**

BSD/OS 3.0. The bandwidth of the link between the client and router was limited to 1.5 Mbits/sec, using a network emulator driver written by Venkata Padmanabhan. The Web client was running Surge, a HTTP request generator written at Boston University [1]. The Web server was running Apache 1.3b5. The Web server document pool consisted of 200 objects with sizes from 198 bytes to 700 kbytes with a distribution in sizes described in [2]. Characteristics of the request pattern (in particular, document popularity, temporal locality of requests, embedded document count of HTML pages, and length of a client's active and inactive periods) all follow representative distributions reported in [2]. We made approximately 75% of the documents negotiable by creating Type Map files for them and alternate representations of the documents. With this partition of negotiable and non-negotiable documents and the access pattern requested by Surge, 60% of the bytes transferred were from negotiable documents and 68% of the documents transferred were negotiable documents, both agreeing with characteristics from client-side traces [6]

To stress the Web server, we started with an initial fixed client population requesting objects from the Web server. Every 800 seconds, additional clients were added to the client population by restarting the Surge process on the client machine. This continued until a maximum number of simultaneous clients were downloading objects from the Web server. We measured the traffic on the link between the client and router and the throughput of the server as a function of time under four scenarios:

- An unmodified Apache server with an unconstrained network link.
- An Apache server modified for content negotiation with an unconstrained network link.
- An unmodified Apache server with a network link constrained to 1.5 Mbits/sec.
- An Apache server modified for content negotiation with a network link constrained to 1.5 Mbits/sec.

Comparing the first two scenarios allows us to quantify the overhead of our content negotiation implementation. Comparing the second two scenarios allows us to quantify the benefits of content negotiation in increasing throughput as a large number of clients access the Web server through a bottleneck link which is at or close to 100% utilization. For the first two scenarios, we started with an initial client population of 5 clients and added 25 clients every 800 seconds until we reached a maximum of 480 clients. For the second two scenarios, we started with an initial client population of 5 clients and added 10 clients every 800 seconds until we reached a maximum of 95 clients.

### 5.3 Results

#### 5.3.1 Overhead of Content Negotiation

Figure 11 shows the overhead of the content negotiation process. The x axis of each figure is the elapsed time since the beginning of the trace, and the y axis is the smoothed throughput of the Web server (measured in connections per second). The periodic dips in the graph are due to the time it takes to restart the Surge process

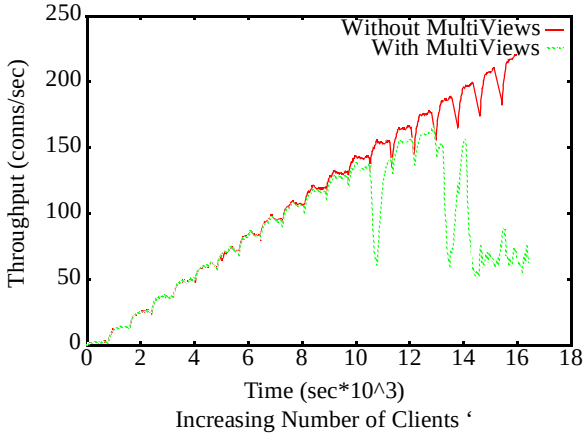


**Figure 11. Unconstrained Apache throughput with and without MultiViews**

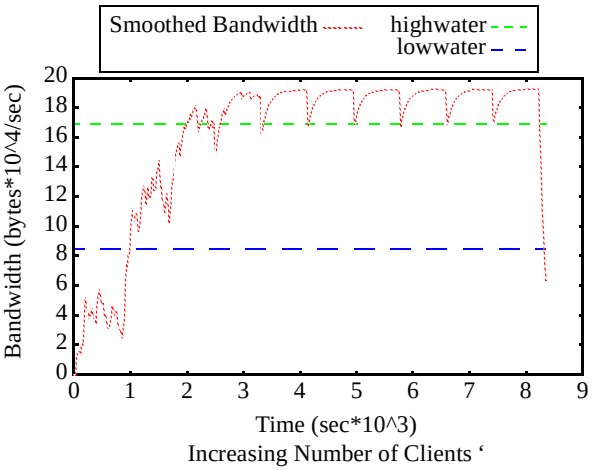


**Figure 12. Bandwidth leaving Apache as a function of number of Web clients without content negotiation**

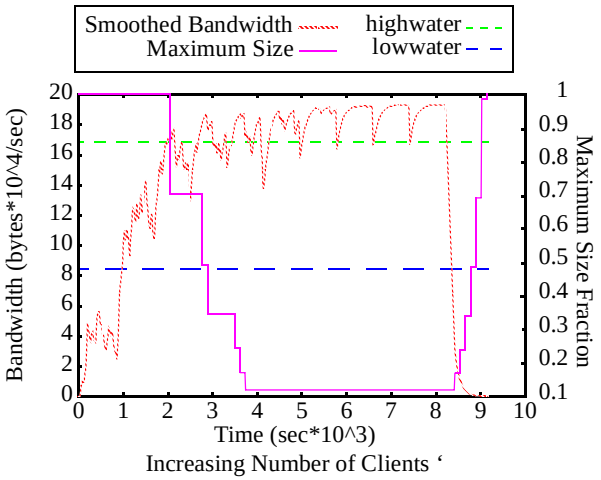


**Figure 13. Bandwidth leaving Apache as a function of number of Web clients with content negotiation**

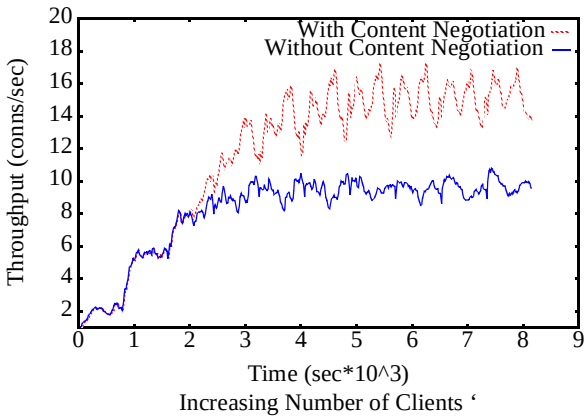


**Figure 14. Apache throughput with and without content negotiation**

with a greater number of clients. There are two curves on the graph. One is for an unmodified Apache server, and the other is for Apache server modified to use MultiViews to serve Web objects. We see that in the early part of the trace (which corresponds to a small number of clients accessing the server), there is virtually no difference between the two curves, indicating that there is little overhead in using MultiViews to serve objects. As more clients are added, however, the MultiViews version of Apache performs significantly worse than the version without MultiViews, and actually performs worse than MultiViews Apache under lower load. From examining the Apache log files, we found that this degraded performance is due to a limit on the number of open files in the system. The MultiViews version of Apache must open additional files (in particular, the Type Map files), and under heavy load, there are not enough file descriptors available to process all client requests, leading to a large number of aborted transactions. We feel that this problem is largely due to the way in which Apache handles MultiViews searches and could be overcome if more effort were put into optimizing the MultiViews process (for example, by caching the contents of Type Map files in memory instead of accessing them from the file system). If this were done, then there would be little or no overhead incurred by using Content Negotiation on a server.

### 5.3.2 Benefits of Content Negotiation

Figure 12 shows the smoothed link bandwidth *sbw* on the connection between Apache and the router as a function of time in the case without any content negotiation. Remember, every 800 seconds additional clients are added, so the number of clients accessing the server increases with time. Also included in Figure 12 are the 90% and 50% values of the limit bandwidth (1.5 Mbits/sec). We see that the bandwidth quickly grows to the limit bandwidth and stays there, implying that the constrained network link is the bottleneck of the system. We can more directly see this effect in Figure 14, which shows the throughput as a function of time. The throughput is limited to approximately 10 connections per second.

Figure 13 shows the smoothed link bandwidth *sbw* on the connection between Apache and the router as a function of time in the case with content negotiation. Also included are the changes in the Maximum Size fraction as a function of time. The bandwidth still approaches the maximum link bandwidth, but because the server can reduce the size of objects being served, it approaches the maximum more slowly. This leads to a greater peak throughput--approximately 15 connections per second (Figure 14).

Although using content negotiation does increase the peak throughput of the system, the increase is rather modest (a 50% increase). This is mostly because only 60% of the bytes come from negotiable documents. The bandwidth requirement of the non-negotiable documents is unaffected by content negotiation and becomes a larger fraction of the total bytes transferred as negotiated documents become smaller.

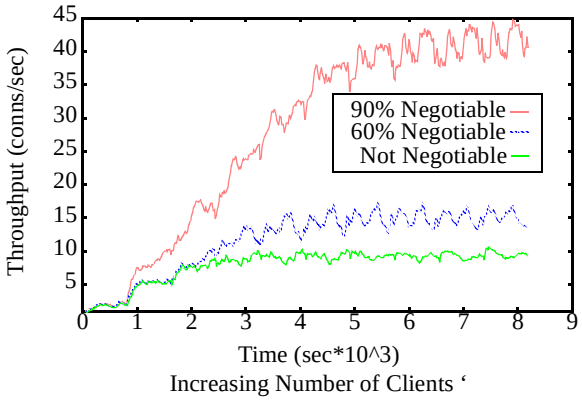


**Figure 15. Apache throughput for varying fractions of negotiable documents.**

The choice of 60% was from analyzing a client-side Web trace and as a result captures the average characteristics of a large number of Web servers as compared to the specific characteristics of a particular Web server. Some Web servers may serve out a higher or lower fraction of negotiable bytes than 60%. For example, a sampling of the pages at the New York Times Web site on March 28, 1998 indicates that approximately 90% of its bytes are negotiable. In this case, the benefits of server-side content negotiation will be more significant in handling additional client load.

To quantify this, we repeated our experiment choosing a different set of non-negotiable objects such that 90% of the bytes served by Apache were negotiable. To further stress our implementation, we added twice as many clients every 800 seconds as in the 60% case. The results of this experiment are shown in Figure 15, which shows the throughput for the version of Apache without content negotiation, the version where 60% of the bytes are negotiable, and the version where 90% of the bytes are negotiable. The 90% version of Apache has a peak throughput four times the peak throughput of Apache without content negotiation, because it can use its limited bandwidth resources across a larger number of clients.

We can conclude that there is potential to dramatically increase the throughput of a Web server by using content negotiation, but the actual benefits are very dependent on the fraction of bytes that are negotiable.

## 6. Conclusions and Future Work

In this paper, we have examined the effectiveness of using HTTP content negotiation to improve the response time observed by users. Web clients that use content negotiation can achieve a dramatically reduced download time as compared to clients that do not use content negotiation. A significant fraction (60%) of Web pages that would normally take more than 10 seconds to download can be downloaded in under 10 seconds using content negotiation. The median transfer time of these Web page drops from 16 to 6 seconds using content negotiation. Web servers can use content negotiation to reduce bandwidth requirements under periods of heavy request load, increasing the throughput of the Web server. This increase in throughput varies from 50% to 400% and is very dependent on the fraction of bytes that come from negotiable documents.

There are several ways in which we plan to further explore the idea of HTTP content negotiation:

- The server-side results in Section 5.3 use file sizes and access patterns generated by Surge, which are representative of Web servers in general but not of any specific Web server. We plan to use traces of accesses to several popular Web servers (such as CNN, the New York Times, and the IBM Olympic Web Server) to examine the benefits of content negotiation for these specific popular Web sites.
- We plan to experiment with different more realistic models for HTTP data transport to determine if a more Web-specific SPAND Performance Report (including portions of the total download time such as DNS lookups and the effect of simultaneous parallel connections) would improve the ability to achieve constant download times with variable quality.
- We also plan to experiment with more conservative use of the SPAND Performance Reports, such as using the 25% quartile of the throughput and round trip time instead of the median throughput and round trip time. This would lead to more aggressive reductions in quality but a greater likelihood that Web pages could be downloaded within the target time interval.

## 7. Acknowledgments

Thanks go to Venkata Padmanabhan for the network emulation driver, Paul Benford for Surge (the client request generator), and to the GloMop group for the image transcoding programs. Thanks also go to the network administrators at IBM Watson for help in capturing the packet traces from clients at IBM research.

## 8. References

[1] P. Barford. surge – Scalable URL Reference Generator. http://www.cs.bu.edu/students/grads/barford/Home.html, 1998.

[2] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Proc. ACM Sigmetrics ’98*, 1998.

[3] N. Borenstein and N. Freed. *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, Sep 1993. RFC 1521.

[4] A. Fox, S. Gribble, E. Brewer, , and E. Amir. Adapting to Network and Client Variability via On-Demand Dynamic Transcoding. In *Proc. 16th ACM Symposium on Operating Systems Principles*, 1996.

[5] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Cluster-based Scalable Network Services. In *Proc. 16th ACM Symposium on Operating Systems Principles*, October 1997.

[6] S. Gribble and E. Brewer. System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. In *Proc. 1997 Usenix Symposium on Internet Technologies and Systems*, December 1997.

[7] K. Holtman and T. Hardie. Content Feature Tag Registration Procedure. http://genis.win.tue.nl/ koen/conneg/draft-ietf-http-feature-reg-03.txt, November 1997.

[8] K. Holtman and A. Mutz. Tranparent Content Negotiation in HTTP. http://genis.win.tue.nl/ koen/conneg/draft-ietf-http-negotiation-06.html, January 1998.

[9] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker. Agile Application-Aware Adaptation for Mobility. In *Proc. 16th ACM Symposium on Operating Systems Principles*, Oct 1997.

[10] S. Seshan, M. Stemm, and R. H Katz. SPAND: Shared Passive Network Performance Discovery. In *Proc. Usenix Symposium on Internet Technologies and Systems (USITS ’97)*, Dec 1997.