Title
Automatically Downloading Images to Improve Web Transfer Times

Authors
Chandranmenon, Girish
Varghese, George

Publication Date
2001-09-11

Peer reviewed

Automatically Downloading Images to Improve Web Transfer Times

Girish P. Chandranmenon
Bell Laboratories
girishc@dnrc.bell-labs.com

George Varghese
University of California, San Diego
varghese@cs.ucsd.edu

Abstract—
Advances in technology have led to the introduction of Gigabit and even Terabit links. However, latencies are limited by the speed of light. In this paper, we propose a technique called Auto Inline Download, which enables servers and clients to transfer a web page and its associated embedded data (such as images, applets etc.) using just one GET request. This saves overall retrieval time by eliminating the delays associated with buffered request processing, and saves a round trip time if the inline data reference is in the last segment of the data transfer for the web page itself. During our experiments, we also found that with the current segment size of 512 bytes, TCP will take at least 7 round trip times to transfer an average web page, with its images, due to its slow start mechanism. This implies that even when we have infinite speed links, TCP slow start will limit the transfer of a web page to at least 490ms assuming a coast-to-coast latency of 70ms.

The bandwidth of a transmission technology is the number of bits the transmission technology can carry per second, while the latency is the time it takes for it to transfer one bit between the transmission endpoints. In order to minimize the overall transfer time, we need to minimize both the bulk transfer time, as well as the latency of the first bit. Although network technology has improved steadily over the past several years, the improvements have been primarily in carrying more bits per second over a wire (i.e., improved bandwidth), rather than transferring a single bit faster (i.e., improved latency).

We argue that once bandwidth is plentiful, the round trip times taken in protocol handshakes will dominate the overall transfer time. Our research focuses on reducing the number of such round trip times spent in transferring a webpage. In this paper we describe a mechanism that should improve the overall transfer time for a web page, and its associated images.

Our mechanism, *automatic download of images* allows the servers to send all the documents (e.g., images) required to render a page without waiting for explicit client requests for each. It improves transfer times in two ways: First, in a typical implementation of a web client (e.g., libwww client from w3c), the client would try to collect as many requests as possible before sending it to the server, thus avoiding too many small writes to the operating system. This buffering mechanism is controlled by a timer. By eliminating the explicit client requests for inline data, our idea also eliminates the timers and delays associated with the buffered request processing. Second, it also eliminates one round trip time, if the image is in the last segment of the data transfer for the HTML file.

In order to evaluate our ideas, we implemented them in a publically available server (apache) and client (libwww) software and measured the performance. In these experiments we found that the average savings was 49ms (23.5%), the timeout value for the buffered request processing stream. We also collected the web graph of our university 4 levels deep starting at the university's home page, collecting 4780 documents, of which 1055 were HTMLs. This graph enabled us to estimate the impact of our ideas, based on the distribution of our university's graph.

The most surprising finding during our experiments was that TCP slow start will take 7.2 round trip times to send an average web page. This implies that even if have a Gb/s link between a server and a client, if the round trip time is 70ms, we cannot receive an average web page with its images (total size of around 40K), in less than 490ms, using a TCP that uses a 512 byte segment size, if we increase the segment size to 1460, the number or round trip times go down only to 5.8 round trips. Thus we believe TCP's slow start mechanism will be the bottleneck in the future.

In Section I, we introduce and discuss the idea of downloading inline data automatically, and discuss why it can be beneficial. Section II discusses our implementation strategy and results. Section III explains how our downloading idea interacts with TCP congestion control, and provides a model for assessing the benefits of our mechanism, now and in the future. Section IV summarizes the paper.

I. Avoiding Multiple Requests

Using current web protocols (HTTP/1.0, HTTP/1.1) to retrieve pages embedded with images and other files, a client has to send one request for the document and one request for each secondary reference. Since the client does not know the names of the secondary files in a document until it receives the base file, it cannot make requests for those until the base level file has arrived at the client. In the case of frames and applets the scenario can get worse: it is only after some frame or applet A referenced in a page is fetched that the client can know that A has more secondary references, such as B, C, and D that also need to be fetched. This process can repeat since B could depend on E and F, and so on.

Newer network technology with large delay bandwidth product pipes make round trip times more expensive. Since most web pages are small to medium size, and most requests are small units of data, a typical web transaction between a browser and a web server can consist of long periods of waiting interspersed with small periods of data transfer in both directions. This is very harmful to TCP, since TCP is optimized primarily for unidirectional continuous file transfer, and not for intermittent and short file transfers.

Auto Inline Download (AID) is an attempt to improve overall transfer time by allowing the server to send the secondary references as soon as it has finished sending the base page, without explicit requests from the browser. In the rest of this paper, we describe and evaluate the idea and experiments with respect to images. However, please bear in mind that this ideas is applicable to all inline data including *frames, javascripts, applets, and image maps* besides *inline images*. The idea of downloading all images was briefly suggested in [PM95] but was not explored

Work done while at Washington University, St. Louis

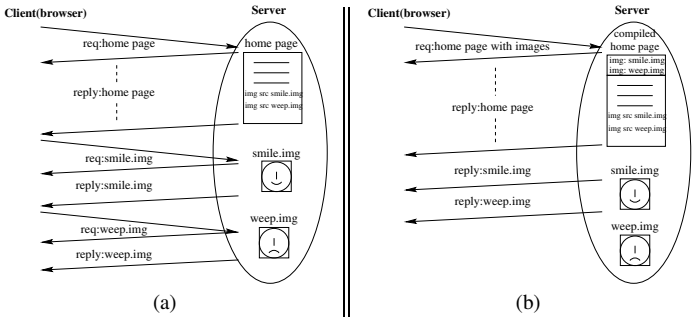

Fig. 1. Auto Inline Download: (a) shows a typical web transaction without AID, and (b) shows a transaction with AID. The round trip times saved become more significant when the images themselves are small. Most images are icons of various types, many of size less than a few hundred bytes.

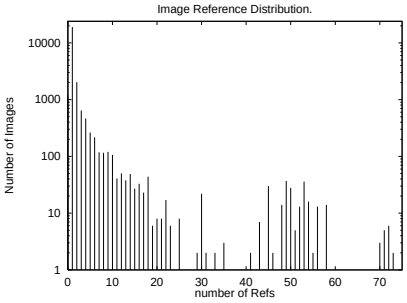

Fig. 2. Distribution of Image References: a vertical bar at x=i indicates the number of images with reference count i. Of the 23782 images analyzed were 22 images with reference count larger than 75. and the largest one had a reference count of 360.

further.

The example in Figure 1 illustrates our idea. In the normal case, Figure 1(a), the client has to make a number of requests to get all inline images. Even if we can send all the requests for inline images at once, the overall time spent in retrieving the images may be more than the time it should take to transfer the images, due to the round trip times involved. These waiting periods can be avoided altogether, if the server is able to send all the images the page needs immediately following the page, as shown in Figure 1(b).

A. Interaction with Image Caching

One could argue that the server should not unilaterally send all images without the client's explicit request, since they could be wasteful in two ways: first, the client may not want any images as is true for many home users who turn off images in their browser; and second, the images could possibly be cached.

We believe that the first problem is not that serious; especially since higher bandwidths, bandwidth saving image formats such as PNG, and the use of cascading style sheets have made it possible for people to download pages in full. Besides, most modern pages are impossible to navigate without the images. Just in case, someone needs to turn the images off, we can still add a flag to the GET request to stop the server from sending them.

The second problem, that of redundant image transmision, needs more careful evaluation. Figure 2 shows the distribution of image references for the Washington University web graph that we collected. The x-axis shows the number of times an image is referenced, and the y-axis shows the number of such

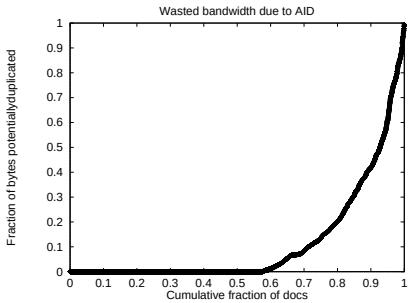

Fig. 3. Worst Case Bandwidth Wastage: Assuming an extremely conservative worst case scenario of 'if an image is referenced from more than one document, it is cached', we compute the fraction of bytes transmitted unnecessarily. X-axis is the cumulative fraction of documents, and y-axis shows the wasted percentage of bytes.

images — i.e., the vertical bar at x=1 shows the number of images referenced exactly once. Of the 23782 images examined, 19046 (80%) are referenced exactly once. A small number (22) of images with reference counts larger than 75 are left out of the graph, since their reference counts were sparsely distributed and cannot be easily depicted. The largest reference count was 360. These numbers suggest that image replication should be relatively infrequent.

Even if a large percentage of images are referenced more than once, the amount of wasted network bandwidth caused by these images depends on the size of these images with respect to the HTML files that reference them. To quantify the wastage of bandwidth, we performed the following experiment.

For each document in the web graph, if an image embedded in it is referenced more than once in the entire graph, we assumed that that (in the worst case) the image would be cached, and sending it from the server without a request from the client would result in wasting bandwidth equivalent to the size of that image. With this assumption, we calculated the wasted bandwidth as a fraction of the total size of the document for each document in the web graph.

More precisely, we calculated the wasted bandwidth for each document as

$$\frac{(\sum \text{sizes of images with reference count} > 1)}{\text{total size of the document including all images}}.$$

This is shown in Figure 3. Notice that this measure is quite conservative because it assumes that every access to every replicated image is redundant. It seems unlikely that access patterns will conspire to have users frequently access pages that share an image in a short period of time.

Using this conservative measure, 57.6% of the documents will not waste any bandwidth due to caching, if we send the images without an explicit request from the client. Clearly, this is because all their images are referenced only by themselves. On the other hand, there are some documents which can cause an overhead of almost 100%. These are documents which contain a small description of an image, and an embedded image that is referenced more than once. One such example (the document with the largest overhead in our sample) is http://www.physics.wustl.edu/mcdonnell/cover.html which has a size of 613 bytes, and contains an image cover.gif of size

110678 bytes. It is promising to note that such cases are only a small percentage of the overall collection.

We observe that 80% of the images are referenced exactly once, and 57.6% of the pages do waste any bandwidth due to caching. The average wasted bandwidth, assuming every page is accessed with the same frequency, is 11.4%. Note that the actual wasted bandwidth could be higher if the access frequencies of pages that contain replicated images is higher than those of other images.

In summary, preliminary measurements show that the wasted bandwidth caused by interactions of AID with image caching should be small. We also note that (increasingly) many pages contain dynamically generated images (e.g., advertisements) and are not allowed to be cached in the client. This trend should further reduce the redundant bandwidth caused by AID. Finally, AID also can also be used for documents with frames. Frames have to be fetched in order to render the page, and it is unlikely that the frames of a document will be cached without the document itself being cached.

II. Integrating AID into the web

Implementing AID necessitates the server to know the contents of a web page, at least the list of its embedded documents. In today's web, the server will have to parse the page to find the references to inline images, and send them. Since it is prohibitively expensive to have a server parse every web page it serves, we suggest preprocessing the web pages to add MIME headers that contain information about all the secondary references in the page.

We propose prepending a MIME header `Img-File` that lists all the images in the web page. A client making a request to the server for a web page will send an additional header `Send-Images` (in order to inform the server whether it should send any images at all). When a server receives a request with such a header, it will send to the client all the images listed in the `Img-File` header at the start of the web page. Since the MIME header is at the start of the page, the server does not have to parse the entire page to find the list of images.[1]

We implemented AID (only the simple version) under a testbed that used libwww5.1b[2] based client and the apache1.3a1[3] server. Required modifications for the server and the client were very small. We measured the transfer times for a collection of 24 webpages. The composition of these pages, and the exact details of the experiment are in [Cha99]. We found that Auto Inline Download always reduced the overall time for accessing a document. In our experiments, the average gain (reduction) in latency was 49ms (23.5%). The gains were much smaller than we expected. This was due to its interaction with TCP, which is the underlying protocol used by HTTP. This motivated us to do a detailed analysis on the effects of TCP. (Section III).

[1] We have also identified a a couple of more ways to reduce the chances of server sending images unnecessarily. Due to lack of space, we have omitted them in this paper; interested readers plesae refer to [Cha99].

[2] http://www.w3.org/Library/

[3] http://www.apache.org/

III. TCP Interaction

Why did AID not perform upto expectations? We used the packet traces collected using `tcpdump` to analyze the data transfer. Figure 4 illustrates what really happened. We had expected the server to be able to send all the image data as soon as it has finished sending the base level HTML page. However, the data transfer using TCP progresses at a much slower pace.

This is because TCP, at the beginning of a transfer does not send all the data available at once. It opens its congestion window using slow-start [Jac88], starting at 1 segment, increasing exponentially until a loss. Thus, the second and third packets do not get sent until the acknowledgment for the first one is received. Therefore, even if the server side HTTP processing has handed the image data to the TCP processing in the kernel, the data cannot be sent until several round trips are over.

One may think that this problem is because the HTML page is fetched on a new connection. However, even with the use of HTTP/1.1 which advocates persistent connection — i.e., use of the same connection for multiple transfers from the same server — the problem still persists. TCP goes back to slow start as soon as the connection has been idle for the duration of a retransmission timout (starts at roughly 1.5 seconds), as estimated during the previous non-idle period. Even with persistent connection HTTP, after one file has been transferred with images, the next file will be requested at the next user click. It is almost certain that the wait before a user click will be more than a retransmission timeout, thus causing TCP to return to slow start.

A. A model of TCP Interaction Effects

Although our savings over a small sample of real experiments were small, we wanted to analyze the interaction of TCP slow start and AID further. In the following paragraphs we propose a model and evaluate the number of round trips required to transmit a file with and without the use of AID. This provides us with a rough characterization of a document that can benefit from AID in terms of its size, and the underlying packet sizes that TCP uses.

We assume the following: the link between the server and the client has infinite bandwidth, both the server and the client can do all processing in zero time and the entire HTTP transfer takes place in the slow start phase of TCP. Thus our model assumes that the only component of latency is the number of round trip delays. Since the slow start phase of TCP starts off at 1 segment, exponentially increasing its window on every round trip, the number of round trips required to transfer a data segment of length ℓ bytes using a TCP segment size of p bytes is

$$\lceil \lg(\lceil \ell/p \rceil + 1) \rceil \tag{1}$$

(e.g., for a data transfer of 1 segment it takes 1 round trip, for 2-3 segments it takes 2 round trips, for 4-7 segments it takes 3 round trips and so on.)

Whether or not AID saves a round trip delay, depends on where the last reference to an image in the original document is. If the reference to the last image arrives at the client early in the document, and at least one extra round trip is required to transfer the remainder of the data needed for the document (remainder of HTML file and images), then adding an explicit

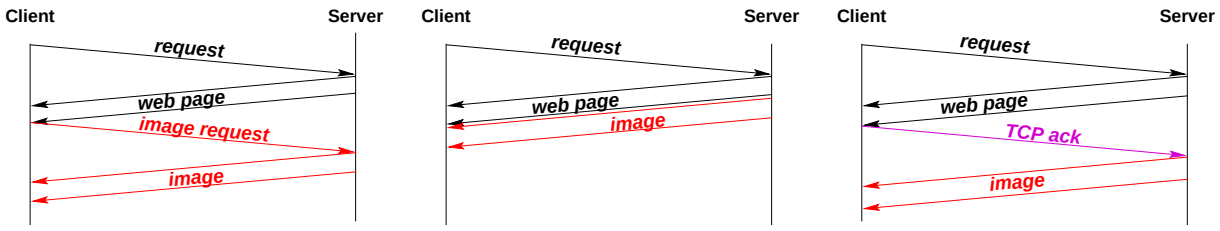

Fig. 4. Interactions of AID with TCP: Although the server has passed the image data to the kernel, it is not sent to the client until after the acknowledgment for the web page itself is received. During this time the client can send the image request concurrently with the acknowledgment for the web page, thus nullifying the savings of AID.

request for the image will not cause an additional round trip. On the other hand, if the reference to the last image arrives towards the end of the document and the remainder of the document (including images) does not require an additional round trip, then AID will save a round trip delay over the normal mechanism.

To make this precise, let x be the offset of the reference to the last image from the start of the base document and let S be the total size of the document including all images. The client will know about the last image reference in $i = \lceil \lg(\lceil x/p \rceil + 1) \rceil$ round trips from the start of the request for the web page. This means that the client can send an image request to the server and receive the image (using the normal mechanism) within one more round trip time. In other words, the server can respond to a request for this image from the client in $i+1$ round trips, even without the use of AID.

Let $k = \lceil \lg(\lceil S/p \rceil + 1) \rceil$ be the number of round trips required for the server to send all the data including the images to the client, using AID. If $k \geq i+1$ AID will not save any round trips in the data transfer. Also, k can never be smaller than i. Therefore, in the case where $k = i$, AID will save one round trip.

Note that this criterion depends on the TCP segment size p. In what follows, we evaluate when this condition can be satisfied for various values of p, using the documents from our web graph.

For each document in the graph, we computed two numbers: the number of round trips it takes the client to learn about the last image reference in the file (i), and the number of round trips it takes the server to send all the data, including all the images to the client, without any request from the client (k). k is the number of round trips required with AID. k and i are calculated using equation 1.

When $k = i$, AID saves one round trip. Figure 5 shows the improvement in the 1055 documents we analyzed using AID for several packet sizes. As we can see, there is improvement only in a few documents (less than 2%) for packet sizes smaller than 16 KBytes. In order to save 50% of the overall transfer time for 70% of the documents, we need a TCP segment size of 256 Kbytes! The problem is that most documents are large enough to force TCP slow start to take several round trip times.

To quantify this effect, we estimated the average number of round trips required to transfer an average web page with its images (total size about 40K[4]), using several packet sizes. Using a packet size of 512 bytes it takes 7.2 round trips to transfer a document, and using the popular Ethernet packet size (TCP packet of 1460 bytes), it takes 5.8 round trips. Thus we believe that TCP is the limiting factor in the overall transfer time, especially over very high bandwidth links.

These results indicate that with the current version of TCP and current segment sizes, AID will not provide dramatic improvements in latency because current documents require several round trips anyway. However, our results also indicate that if segment sizes increase sufficiently that AID can reduce latency from two round trip delays to one round trip delay. Alternately, it can be argued that while TCP Slow Start is extremely successful at combating congestion, there is a need for a different version of TCP (perhaps use rate based congestion control) that reduces the number of round trip delays required for documents over high bandwidth (and large latency) links.

B. *When will AID be useful?*

In the previous subsection, we saw that using the current version of TCP, standard segment sizes, and typical Web documents, AID can reduce round trip delays (by at most one) for only 2% of the pages. However, in our experimental evaluations we saw that we had 49ms savings on the average, even with a very small round trip delay on an Ethernet (around 1 or 2ms). How can we reconcile these two seemingly conflicting statements?

The explanation of this apparent anomaly is that our model assumes that round trip delays are the only component of latency. However, in the Ethernet experiment, the delay between finding an image reference and sending the actual request happens to be significant (around 50 ms) and this delay is avoided by AID.

Thus savings are caused by the implementation of client side requestor. In `libwww`, once the base HTML page is being downloaded, the client side collects further requests (typically image requests) in a buffer, until there is a packet's worth to send to the server. This is done for two reasons: first, the client side wants to avoid small writes to the kernel, and thus avoid the generation of small packets on the Internet. Second, the server side will benefit from large reads, instead of one request at a time from the server kernel. The delay added to flush this buffer at the client side is 50ms. We believe that this is the time we saved in our experiments, since we do not send any request at all to the server.

While the reader may feel that this improvement is specific to the implementation we used, it is likely that any client im-

[4]This was based on the measurements from [AW96].

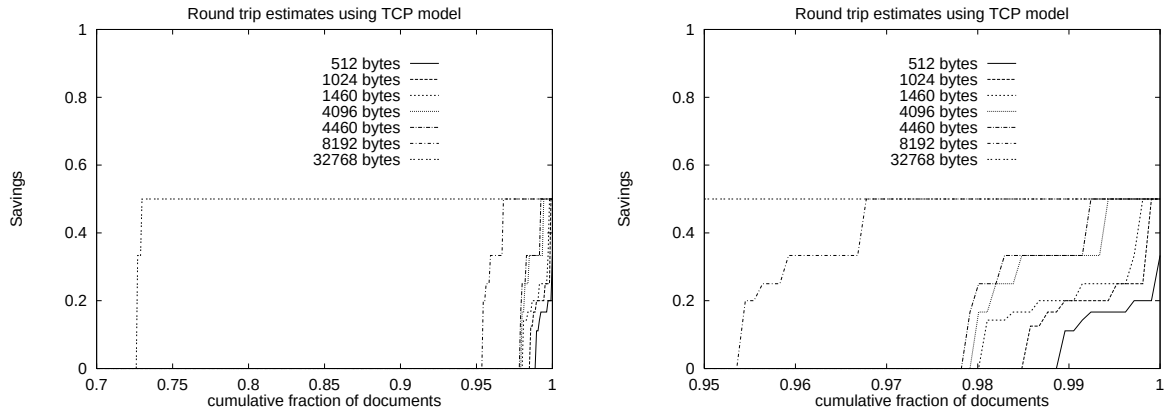

Fig. 5. Estimated Savings of Round Trip Times: The x-axis is the cumulative fraction of the documents, and the y-axis is the fractional savings in round trip times. Note that the maximum savings is 0.5, in the case where the HTTP transfer takes 2 round trips, and AID saves 1 round trip time. For the part of the graph x=(0..0.7) y = 0.0, i.e., there was no gain in number of round trips.

plementation would use a similar mechanism to improve performance. There is a basic conflict between making image requests as soon as a reference is detected (to improve latency) and batching small requests (to improve throughput). This tradeoff is avoided by AID. Alternately, the `Img-File` header can help the client batch requests without resorting to a delay timer.

Thus we believe that the usefulness of AID today comes not from saving round trips, but saving server and client side processing. This was a somewhat surprising discovery for us. We leave a more detailed evaluation of this effect for future work.

Even if AID is not useful at some level of technology, we believe the `Img-File` header is still useful. If this header is sent to the client, it enables the client to make image requests early, without waiting for the HTML page to arrive, which may take several round trip times, to arrive and be parsed.

IV. Summary

In this paper, we proposed and evaluated the idea of sending the images, applets and other inline data to the client without an explicit request for each of such documents. Although we had expected significant reduction in overall retrieval time because of the elimination of explicit image requests, we did not see as much improvement in the experimental evaluation. Further studies revealed that interaction with TCP forces explicit requests to overlap with the transfer of the base HTML file itself, and thus the explicit requests do not often cause any extra waiting period. The actual improvement appears to be a result of avoiding a request buffering timer and not a result of reduced round trips.

Despite this, we make the following observations about the potential utility of AID. First, in a pathological case where the image file reference is at the end of an HTML file, the request from the client can force a waiting period at the server. Therefore, the more modest idea of collecting all the image names as a header can be a good idea. This can help the client fire off all the image request as soon as the start of the web page arrives at the client. Second, in the future when TCP slow start is not the dominant congestion control protocol, AID may be much more important for HTTP transfers. Third, AID can be useful for more complex documents that consists of frames with multiple levels of nesting.

The AID measurements taught us another lesson. TCP slow start is a major cause of extra round trip delays because the ratio of the size of a web page to the TCP segment size is fairly large today. For example, our model and measurements indicate that the average web document takes 7 round trips using the standard network segment size of 512 bytes. This problem can be solved using larger segment sizes. However, increases in segment size only result in logarithmic reductions in round trip times. For example, a number of TCP implementations are already using 1460 byte segment sizes; our measurements indicate that the average web document takes around 6 round trip times using a 1460 byte segment size. Thus, we believe that designing a new version of TCP that can handle congestion but can also allow high initial throughput (at least in the absence of congestion) is an important research problem.

This experience teaches us an important lesson: an idea that makes perfect sense by itself may not work as well when incorporated into an entire system because of interactions with other parts of the system. This interaction could be as simple as discovering that the part to which our idea applies is not a significant bottleneck for system performance. It can also be that the idea itself is excellent, but the idea cannot be incorporated into the system because it is not backward compatible, and the system has a widely installed base which cannot be changed overnight.

Neither of these two standard problems afflicts AID. The real problem is that it interacts undesirably (in terms of performance, not correctness) with two existing system features: client document caching and TCP slow start. While we believe the former is not a problem, we have shown experimentally that TCP slow start does limit the gains of AID. We have also described a simple model that can predict when (and by how much) AID can provide round trip latency gains in the face of slow start.

To quote Tanenbaum[Tan92]: *the moral of the story is, getting it right in practice is much harder than getting it right in principle.*

References

[AW96] M. Arlit and C. Williamson. Web server workload characterization: The search for invariants. In *In proceedings of SIGMETRICS'96*, May 1996.

[Cha99] Girish Chandranmenon. Reducing web latencies using precomputed hints. Technical Report PhD Thesis. Technical report WUCS-99-18, Dept of Computer Scie nce, Washington University in St. Louis, August 1999.

[Jac88] Van Jacobson. Congestion avoidance and control. *Proceedings of the ACM Sigcomm '88 Symposium on Communications Architectures and Protocols*, 18(4):314–329, August 1988. part of ACM Sigcomm Computer Communication Review.

[PM95] Venkat Padmanabhan and J. Mogul. Improving http latency. *Computer Networks and ISDN systdms.*, 28:25–35, Dec 1995.

[Tan92] Andrew Tanenbaum. *Modern Operating Systems*. Prentice Hall, Upper Saddle River, NJ 07458, 1992.