

The Case for Better Bandwidth Estimation

Brian D. Noble, Li Li, Atul Prakash

Electrical Engineering and Computer Science Department
University of Michigan, Ann Arbor

November 30, 1998

Abstract

A Web proxy must accurately predict network performance between itself and its servers and clients in order to make good distillation decisions. In this paper, we show that the current approaches — either assuming the proxy is well-connected to all servers or using long-term observations — to make such predictions are insufficient. We propose a new prediction method, *estimation with uncertainty*, that will play a crucial role in web proxies as well as in other important problems.

1 Introduction

The diversity of speeds with which end users access network services is steadily increasing. Typical home users have network connectivity in the tens of Kb/s — several orders of magnitude slower than institutional users. Mobile users can experience connectivity that varies over similar ranges as the mobile host takes advantage of deployed network technologies. Web services are often optimized for those users connected by high-speed links; weakly-connected users are then forced to cope with poor performance when accessing those services. Rather than force services to optimize to the slowest potential customer, a proxy-based *distillation* architecture has been proposed [1].

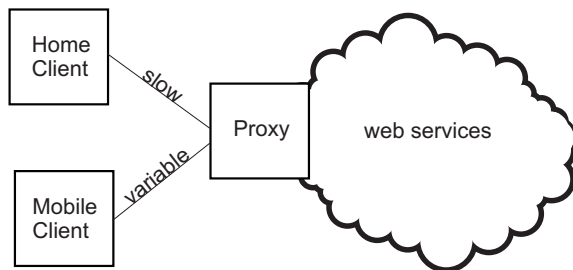


Figure 1: Proxy Architecture

In this architecture, shown in Figure 1, a proxy is placed at the border between weakly-connected clients and the rest of the network. The path from a server to the proxy will often be much faster than that from the proxy to the

client. The proxy can then distill each object fetched from the server into a version that is smaller, but of lower fidelity. When the server-proxy path is faster than the proxy-client path, the overhead of distillation is more than offset by the shortened transmission time along the slow link, improving response time for the client.

Correct distillation requires good estimates of the speed along the server-proxy and proxy-client paths. For example, users connected by mobile links can have distillation decisions made for them *dynamically*. Objects are distilled little if at all when the bandwidth between proxy and client is high, and are aggressively distilled when that path is slow [6].

Unfortunately, proxies often assume that all remote services are *well-connected* to the proxy, and focus only on the speed of the path to the client. A Web server that is connected to the rest of the network by a slow modem line will never serve items quickly enough to justify distillation. More recently, a proxy that uses long-term average behavior to each destination has been implemented [3]. This proxy can discover persistent performance limitations, but cannot predict short-term variations in performance due to transient load.

This paper argues that both of these approaches are inadequate as the basis for making distillation decisions. In order to show this, we have measured how each approach would affect the Web browsing of a small group of users at the University of Michigan. Section 2 describes how these measurements were collected. Section 3 quantifies the impact of assuming that connections are always fast or can be predicted with long-term averages. In Section 4 we propose a new method, *estimation with uncertainty*, that will provide for better distillation decisions.

While we have focused primarily on Web proxies, a good bandwidth estimator is crucial to a number of important problems. For example, one needs good estimates of network performance for prefetching in a distributed system [10, 11]. Good estimates are also essential in distributed query planning [2, 4] and caches that consider the cost of re-fetching in replacement decisions [12].

2 Experimental Method

In order to explore the impact of various assumptions in distillation decisions, we measured the Web browsing of a small set of users in the EECS department of the University of Michigan. This traffic was measured at a proxy for all of our users' Web activity over one month. The proxy provided no distillation or other added services; it was used only to measure fetch throughput from remote servers.

The proxy generates an *access log* that records information about client requests, one line per request. When a request arrives at the proxy, it is forwarded to the remote web server, which provides a response. An HTTP response consists of a set of headers followed by the requested data. The proxy first receives the headers and sends them back to the client before reading and forwarding the data itself.

We have modified our proxy to record two timestamps for each fetch: one after fetching the headers, and one after fetching the end of the data. The log record for each fetch contains: client IP address, server IP address, URL, MIME type, request time, post-header timestamp, post-fetch timestamp, and the number of data bytes received. In order to guarantee privacy — and encourage as many people as possible to use our proxy — we log only the MD5 hashes [8] of the first three entries. Thus, we can correlate fetches to the same server, but cannot identify servers, clients, or fetched objects.

The throughput between the server and the proxy is defined as:

$$T = \frac{b}{t_f - t_h} \quad (1)$$

Where b is the number of data bytes fetched, t_f is the timestamp taken after all data is obtained, and t_h is the timestamp taken after the headers have been fetched.

Browsers fetch a web page by first fetching the page's text and then issuing separate requests for each embedded object. In order to overlap connection set-up and tear-down with the transfer of useful data, browsers often issue requests for embedded objects concurrently. These requests likely are to the same server; an example is shown in Figure 2.

The times t_1 and t_2 denote the periods during which the proxy receives data from the server for the first and second request, respectively. In this example, the transmission of the two responses happens concurrently, and the available network capacity is divided between them. If we were to measure the throughput separately for each request, we would understate the aggregate throughput between the server and the proxy. Thus, whenever two log entries with the same endpoints overlap in time, we merge them into a single entry. The new entry has a size that is the sum of the size of the two merged entries. The first timestamp is the

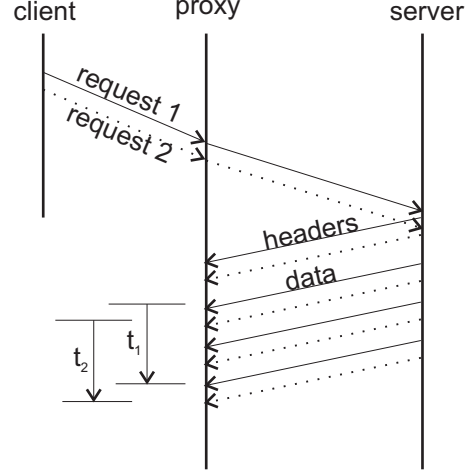


Figure 2: Parallel Requests for Embedded Objects

smaller of the post-header timestamps, while the second is the larger of the post-fetch timestamps. We found that 22% of logged responses experience such overlapping.

The proxy interleaves fetching pieces of an object from the remote server with forwarding those pieces to the client. If the path from the proxy to the client is the bottleneck, the measured throughput will again understate the capacity from server to proxy. We do not log such observations; they are extremely rare, since our users almost always have excellent connections to the proxy.

We modified Apache version 1.3.1 to act as our data collection proxy. It ran on a 300 MHz Pentium II machine with 64 MB of RAM running Linux 2.0.35. The log file was approximately 22 MB with an average of 4000 requests per day. Altogether there are 1660 distinct servers in the log. However, about 40 servers account for more than 50% of all the requests. The logged responses include 76% *successful* responses, 17% *not modified* responses, and 7% other responses. We report only the successful responses.

3 Experimental Results

We used the collected data to answer two questions. First, what is the impact of assuming that the paths between the proxy and all servers are fast? Second, what is the impact of using long-term observations to predict individual fetch times?

3.1 Assuming Fast Links

The most common strategy used by distilling proxies is to assume that all remote services are well-connected to the proxy. In other words, the time for the proxy to fetch an object from a remote server is negligible compared to the time to distill and forward that same object.

Figure 3 shows the distribution of bandwidths observed by our proxy when fetching images from remote servers.

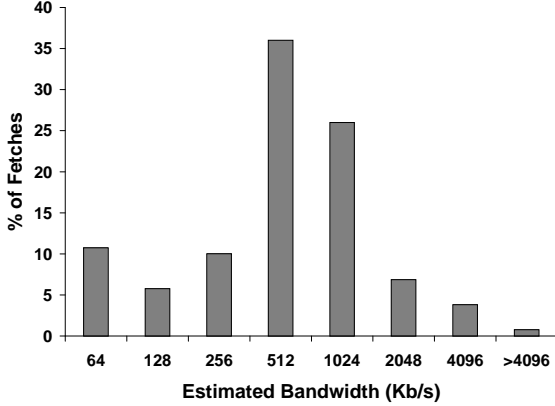


Figure 3: Image Retrieval Bandwidth

Because TCP slow-start can have a dominant effect for transfers over paths with large round-trip times, smaller objects tend to have lower bandwidths. In this histogram, we consider only images that are 5KB or greater in size; most distilling proxies consider only images, and those smaller than 5K in size are not likely to benefit from distillation. This number was selected to be conservative. Other work suggests distilling images larger than 1K [3]; using this size as the cutoff would shift the distribution in Figure 3 to the left.

There were 15,742 images over 5K fetched during data collection. Each bucket represents the percentage of observations that fell between that bar’s X value and the previous one; for example, the second bar shows that almost 6% of all fetches were at bandwidths between 64 Kb/s and 128 Kb/s. The final bucket contains all fetches that were faster than 4 Mb/s.

The central message of this histogram is that more than one in ten images are fetched near or below typical modem speeds. A proxy that assumed all fetches were fast would distill these images rather than simply forwarding them. The user would then see a lower quality image without a corresponding reduction in response time; the server-proxy path is the constraining link, and distillation will not deliver the last byte of the image more quickly. In fact, response time would likely increase due to the added delay of distillation.

3.2 Using Long-Term Averages

Rather than simply assuming that all fetches are fast, the proxy must use some method of estimating the time it will take to fetch an object from a remote service. One suggested approach [3] has been to use a technique similar to SPAND [9] to make these predictions. In this approach, the proxy logs interactions with each server and client in protocol stubs. An off-line algorithm periodically summarizes these observations, grouped by endpoint and size. When each fetch is requested by the client, the proxy con-

sults this summary to predict the time required to obtain the requested object.

The success of this approach depends on the degree to which long-term summaries can predict individual fetch times. To gauge the predictive power of such observations, we first selected the most popular server from our logs.¹ We then predicted the times for a single day’s worth of fetches by relying on the previous week’s fetch times, measuring the degree to which our predictions matched reality.

We used an *error ratio* metric to determine the accuracy of long-term prediction. For the fetch of an object of size s , we first examine the prior week’s fetches to predict its performance. Over wide area links, TCP slow-start dominates the throughput of small transfers; therefore, we consider only those objects larger than $s/2$ and smaller than $2s$. We then take the median of those observations as the predicted fetch time, t_p . We say that this predicted time is correct to within an error ratio, r , of the actual time, t_a , if it satisfies the following inequality:

$$\frac{t_p}{r} \leq t_a \leq r t_p \quad (2)$$

We consider the error ratio of a particular prediction to be the smallest possible value.

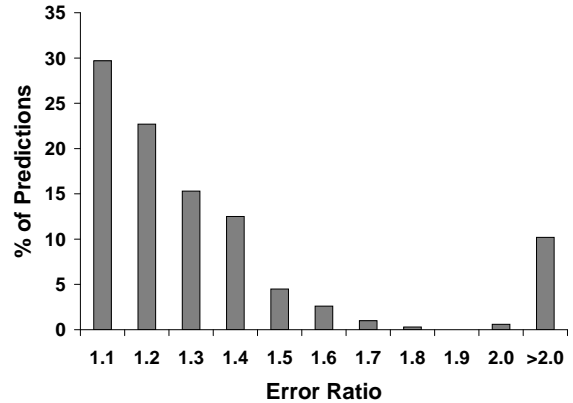


Figure 4: Error Ratio in Long-Term Predictions

Figure 4 shows the distribution of prediction accuracy for fetches of all sizes. There are 313 predictions represented in the histogram. Each bucket represents the percentage of predictions that had error ratios between that bar’s X value and the previous one. For example, the second bar shows that almost 24% of all predictions had error ratios between 1.1 and 1.2. The final bar gives the percentage of predictions with error ratios of 2.0 or greater.

¹ Because we record only the MD5 hashes of server names, we do not know what server this was.

While most predictions are quite good, more than 1 in 10 of them are still off by a factor of 2 or more. Such erroneous predictions can lead a proxy to make sub-optimal distillation decisions. The proxy will either throw away information needlessly or not distill aggressively enough to give good response times.

4 Towards a Better Estimator

In order for a proxy to make the best possible distillation decisions, it must accurately predict each individual fetch. However, it is unlikely that we will be able to build an oracular proxy that is always able to predict correctly. If two hosts are close together — as the client and the proxy are likely to be — performance observations are likely to be very stable. However, traffic performance between two widely separated Internet hosts shows significant variance [7], and observing a sequence of bandwidth observations can lead one to believe they are no more than noise.

In light of this, a proxy should have the following properties to be as useful as possible:

- It must accurately estimate transfer time from servers. These estimates must depend in part on short-term observations.
- Each estimate should be accompanied by a *degree of confidence*. This confidence should ideally be conservative.
- The proxy should take both the estimate and the confidence into account when making distillation decisions. When confidence is low, the proxy should be conservative in the prediction's use unless directed otherwise by user preferences.

As shown in Section 3.2, long-term averages are not sufficient as predictors of bandwidth; the proxy must take into account short-term variations. Simple approaches can work well for the short connection between client and proxy. For example, Odyssey uses:

$$p_i = \alpha b_{i-1} + (1 - \alpha)p_{i-1} \quad (3)$$

where p_i is the predicted bandwidth of the i th fetch, and b_i is the actual bandwidth during that fetch; α is $3/4$, giving a heavy bias towards the most recent observation. This has been shown to have sufficient predictive power in realistic situations [5], while retaining good agility — the ability to react quickly to change in mobile environments.

However, since performance observations over wide-area connections carry much more noise, something more sophisticated is required to predict performance between the proxy and servers. To see why, consider Figure 5; it shows the same predictions as Figure 4, but in different form. Rather than a histogram of error ratios, it shows

the error ratio for the long-term predictions to the most popular server, in sequence. Error ratios above the X axis represent estimates that were too high; those below the X axis are estimates that were too low. Points plotted at the extremes show error ratios of 2.0 or greater.

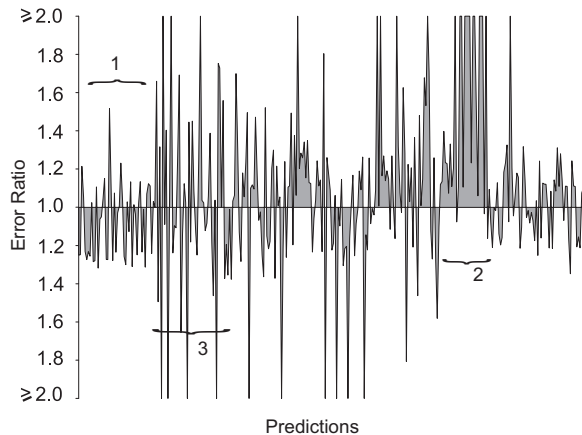


Figure 5: Error Ratio Over a Series of Predictions

On first glance, it appears that there is little predictive power in short-term observations. However, on closer inspection there is some opportunity for improvement. We have labeled three regions of interest in the graph. In region 1, the deviation from the long-term prediction is fairly small; it could be used with a high degree of confidence.

In region 2, the predictions are consistently too high, often exceeding an error ratio of 2.0. During this period, the time to fetch objects from the server was much longer than usual. This could have been caused either by increased cross-traffic, or by load at the server. During this period it would be possible to observe that the long-term average is too optimistic, and quickly adjust the estimates downward to meet reality.

In region 3, the predictions are alternately far too fast and far too slow. If estimates had confidences attached, the proxy would be able to determine that it was in a range of instability. In such cases, it would be best to conservatively assume that the object will arrive quickly at the proxy. If wrong, the proxy might throw away extra information through too-aggressive distillation. The alternative — assuming a lower bandwidth, and forwarding the item unchanged — might result in unnecessarily long response times when the assumption is unwarranted.

We plan to build an estimator that includes confidences to cope with situations similar to the three above. It is depicted in Figure 6, and is similar to a canonical feedback control system. At each fetch, the estimator will compare its predictions with the actual values. By designing an appropriate set of heuristics, one can identify consistent

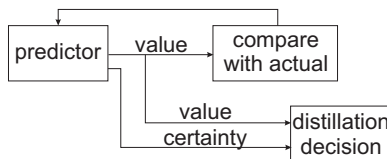


Figure 6: A Better Estimator

errors, such as might happen in region 2, or noisy states, such as region 3.

5 Conclusion

Correct distillation decisions at a proxy require accurate predictions of the time to fetch an object from a remote service. Because the servers storing the original objects are often far away in the network topology, accurately estimating that time is an extremely difficult problem. As we have shown, simply assuming all fetches will be fast is incorrect, as is using only long-term averages and ignoring short-term fluctuations.

We propose to build an estimator framework that uses feedback control to both improve the accuracy of estimates as well as attach a certainty to them. When estimates are accurate, the proxy can make much better distillation decisions. When the estimates are very uncertain, the proxy must make some conservative assumptions in order to preserve good response time for the end users. In addition to distillation proxies, such an estimator would be invaluable in distributed prefetching, query planning, and cache replacement algorithms. We plan to investigate its use in these and other domains.

References

- [1] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *Proceedings of the Seventh International ACM Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, October 1996.
- [2] L. Gravano, C.-C. K. Chang, H. Garcia-Molina, and A. Paepcke. STARTS: Stanford Proposal for Internet Meta-Searching. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, 1997.
- [3] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas. Dynamic Adaptation in an Image Transcoding Proxy for Mobile WWW Browsing. *IEEE Personal Communications Magazine*, December 1998.
- [4] L. Liu and C. Pu. A Dynamic Query Scheduling Framework for Distributed and Evolving Information Systems. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, Baltimore, MD, May 1997.
- [5] B. D. Noble. *Mobile Data Access*. PhD thesis, School of Computer Science, Carnegie Mellon University, May 1998. Available as technical report number CMU-CS-98-118.
- [6] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, St. Malo, France, October 1997.
- [7] V. Paxson. End-to-End Internet Packet Dynamics. In *Proceedings of the 1997 ACM SIGCOMM Conference*, Cannes, France, September 1997.
- [8] R. Rivest. The MD5 Message-Digest Algorithm. Internet RFC 1321, April 1992.
- [9] S. Seshan, M. Stemm, and R. H. Katz. SPAND: Shared Passive Network Performance Discovery. In *Proceedings of the First USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997.
- [10] D. C. Steere. Exploiting Non-Determinism in Set Iterators to Reduce I/O Latency. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, St. Malo, France, October 1997.
- [11] G. Voelker, E. Anderson, T. Kimbrel, M. Feeley, J. Chase, A. Karlin, and H. Levy. Implementing Cooperative Prefetching and Caching in a Globally Managed Memory System. In *Proceedings of the 1998 ACM SIGMETRICS Conference on Performance Measurement, Modeling, and Evaluation*, June 1998.
- [12] R. P. Wooster and M. Abrams. Proxy Caching that Estimates Page Load Delays. In *Sixth International World Wide Web Conference*, September 1997.