End-to-end performance prediction for the Internet (Work in progress)

Richard A. Golding

UCSC–CRL–92–26

June 19, 1992

Concurrent Systems Laboratory
Computer and Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064

Many applications designed for wide-area systems use replication to provide a service at multiple locations. From which site should a client choose to obtain service? A client can specify its needs in terms of communication latency, bandwidth, or error rate. This specification must be matched against the expected performance available from each site. I present the results of two experiments that measured end-to-end performance, and discuss how the results can be used for prediction.

Keywords: distributed systems, communication latency, network bandwidth, performance prediction

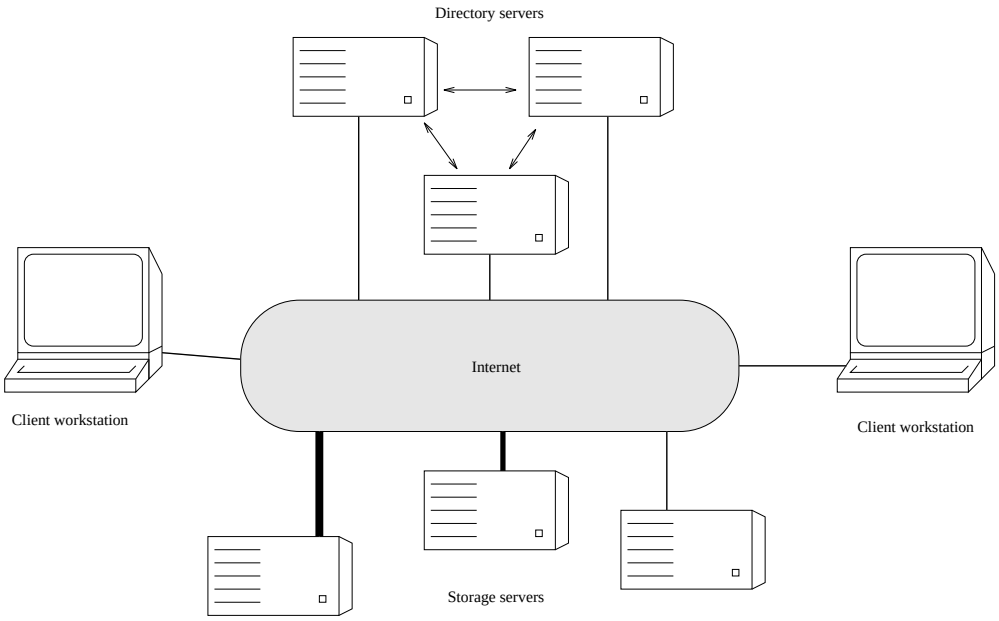

FIGURE 1: Architecture for Internet information services.

1 Introduction

Applications for wide-area systems often use replication to provide a service at several different sites. This approach improves both the fault-tolerance of the system and its performance. In many systems, clients can contact any one server, so that the service is unavailable only when all servers are unavailable. The client can also select the server that will provide it the best service.

In general, the best server for an operation will be the one that will respond most quickly. If the operation requires that only a small amount of information be moved between the sites, message latency will dominate performance. On the other hand, if large amounts of information must be transferred then bandwidth will dominate.

For example, the archie system [Emtage92] provides an index of files that are available for anonymous FTP on the Internet. A user typically first sends one or more query message to archie to find a set of sites providing the file, then uses FTP to transfer it. The query operations typically send only a few bytes to the server and obtain at most a few kilobytes in return, so message latency is most important in selecting an *archie* server. FTP sessions often transfer files of a megabyte or more, so throughput is more important in selecting an FTP site.

I am investigating an architecture for providing replicated information services on the Internet, as shown in Figure 1. In this model, client systems use the information service to locate and access collections of information that I will call "files". This information need not actually be provided as a file. The client software runs on a workstation, and uses the Internet to communicate with directory and storage services. The directory servers process user queries, returning locations and details about files that contain the requested information. A client can then contact storage servers to retrieve the file.

This architecture enables an information service to use several performance-improving techniques. The service can dynamically cache interesting files at several sites throughout the internetwork, improving the distance a client must communicate to retrieve files. Different caching policies can be used depending on the kind of information – a new software distribution system can proactively create many copies, anticipating

future needs, while an information retrieval service can use semantic structure to cache information likely to be used.

I have investigated *quorum multicast* protocols that will use preferred sites [Golding91, Golding92]. These protocols use an ordering on m sites, and attempt to communicate with the best n of them. The sites can be ordered, for example, from lowest to greatest latency.

This approach requires accurate performance predictions. In the next three sections I will consider, first, predicting latency, then using these predictions. I will then discuss a separate experiment on predicting bandwidth. All of this work should be considered preliminary; sample sizes are not always large enough to ensure statistical accuracy and the analysis can surely be improved.

2 Predicting latency and loss

This section is based upon my Master's thesis work [Golding91], which in turn used ideas from Van Jacobson's TCP implementation [Jacobson88]. I was investigating *quorum multicast* protocols, that send a message to a fixed-size subset of a set of sites. The subset is selected according to an ordering on the sites; for example, sites could be ordered by increasing expected response latency. Some of these protocols attempt to handle packet loss through retry. The site orderings and packet loss timers both required accurate predictions of expected communication latency.

I conducted the performance evaluation by collecting samples of packet latency and loss, then using the traces to drive simulation. There were two such experiments, varying mainly in duration. I also built a simple quorum multicast implementation to verify the simulation results.

2.1 Methods

Trace records were obtained by polling the remote host from **maple.ucsc.edu**, a Sun 4/20 workstation in the Concurrent Systems Laboratory (CSL) at UC Santa Cruz. The measurement software was built atop the the ping program, which sends ICMP echo messages. Hosts are expected to respond to ICMP echo messages by returning the message as soon as possible. It polled each host 30 or 50 times (depending on the experiment) every 20 minutes. The first experiment lasted 48 hours; the second lasted one week.

The first measurement experiment used 24 hosts chosen from those hosts with which CSL systems communicated regularly. The experiment collected 50 samples at one-second intervals every 20 minutes for each host, on a Wednesday and Thursday. This resulted in 7200 samples for each host. The hosts, and a summary of their behaviors, are reported in Table 1. One host, **andreas.wr.usgs.gov**, was unavailable for 7 of the 48 hours sampled; the other hosts appear to have been available the entire time.

The second experiment was similar to the first, except that it involved more hosts and behavior was traced over an entire week. For this study I selected 125 hosts on the Internet from a list of several thousand Sun 4 systems. One set of polls was collected for each host every 20 minutes over a seven-day period. Each set of polls consisted of 30 ICMP echo requests issued at one-second intervals. This resulted in 15 120 samples for each host.

2.2 Packet loss behavior

Packet loss is the simplest measure to be obtained from the traces. I first examined the success rate for communicating with each host. The results for the 24-host experiment are reported in Table 1. In the this experiment I found that most hosts would respond to a message more than 90% of the time. The one exception (**andreas.wr.usgs.gov**) represented a host that appears to have been continuously unavailable for 7 of the 48 hours sampled. Combining this information with published results on the reliability of hosts, I

TABLE 1: Hosts selected for first study.

	Location	Mean response latency (ms)	Message success (%)
spica.ucsc.edu	Santa Cruz, CA	0.59	100.00
cs.stanford.edu	Palo Alto, CA	18.50	97.79
apple.com	Cupertino, CA	24.50	95.96
ucbvax.berkeley.edu	Berkeley, CA	24.96	96.43
andreas.wr.usgs.gov	Menlo Park, CA	26.64	79.90
fermat.hpl.hp.com	Palo Alto, CA	39.88	97.92
ucsd.edu	San Diego, CA	51.86	91.15
june.cs.washington.edu	Seattle, WA	52.56	97.11
beowulf.ucsd.edu	San Diego, CA	57.68	93.33
unicorn.cc.wwu.edu	Bellingham, WA	107.88	96.44
gvax.cs.cornell.edu	Ithaca, NY	162.35	95.28
prep.ai.mit.edu	Cambridge, MA	215.97	89.47
lcs.mit.edu	Cambridge, MA	219.13	89.08
vivaldi.helios.nd.edu	Notre Dame, IN	228.96	96.57
acrux.is.s.u-tokyo.ac.jp	Tokyo, Japan	263.68	96.46
swbatl.sbc.com	Atlanta, GA	298.57	97.06
zia.aoc.nrao.edu	Virginia	353.09	97.79
sdsu.edu	San Diego, CA	404.54	92.85
inria.inria.fr	France	1142.99	84.63
top.cs.vu.nl	Netherlands	1312.32	90.42
slice.ooc.uva.nl	Netherlands	1340.40	88.97
cs.helsinki.fi	Finland	1525.78	90.42
mtecv1.mty.itesm.mx	Mexico	1641.70	91.33

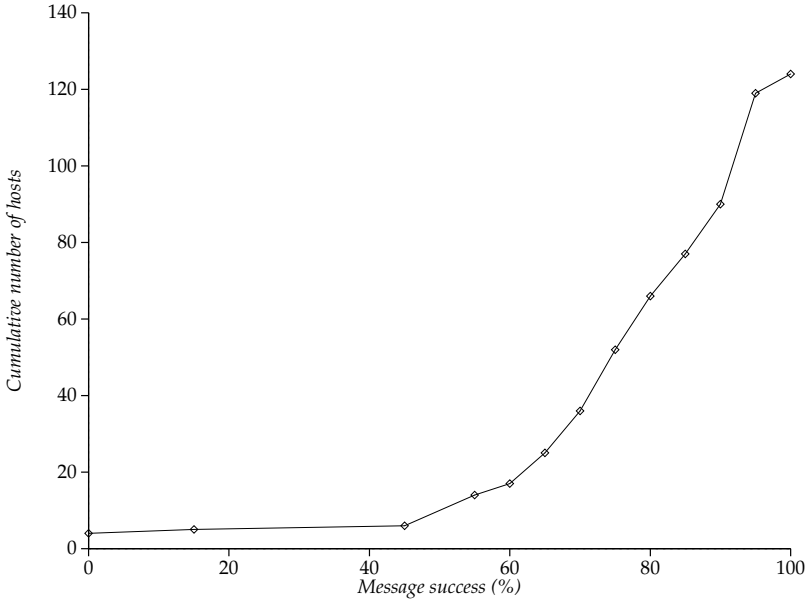

FIGURE 2: Overall packet delivery rates (1-packet loss rate). Average rate 80%.

conclude that communication will succeed most of the time when a host is functioning.

The data from the 125-host experiment are less encouraging. In this set, four hosts were continuously unavailable for the entire seven days, while some hosts exhibited overall packet loss rates of more than 50%. All sample hosts selected were known to exist and function a few weeks before I recorded the traces, and it seems unlikely that these four hosts had been deliberately taken out of service in the interval. The mean loss rate was 20.0%, as compared to 6.7% for the 24-host experiment. Figure 2 shows the fraction of hosts with different overall packet loss rates.

I conjectured that the packet loss rate might be related to the number of gateways that must pass the message. I plotted overall delivery rates against distance (Figure 3.) I had expected that nearby replicas, those that require three or four number hops to reach, would have high delivery rates. Indeed this appears to be the case. However, outside of this local organization the number of hops does not appear to be a good predictor of packet delivery.

Packets can be lost for one of two reasons: they are lost in transmission, or the remote host is down. While host availability cannot be determined exactly, a host that does not answer any pings for 30 seconds is likely to have failed. This is not a perfect measure for two reasons: a gateway or link crash would appear to be a host failure, and because a very busy host could also appear to have failed. Bearing these limitations in mind, I computed an estimate of overall host availability as the fraction of 30-ping data sets containing at least one response to the total number of such data sets collected for the host. Figure 4 shows the distribution of overall availabilities in the 125-host experiment.

Next, I examined the data sets to determine how long communication failures lasted. Failures were classified by the length of the run of lost packets, as shown in Figure 5 and in Tables 2 and 3. The first column in these tables lists those run lengths that contributed to 1% or more of the lost messages. The second column reports the percentage of lost messages that were in runs of each length. I found that in more than half the cases where a packet was lost, it was part of a run of only one or two. The only other significant run length was 30 or 50, the size of one data set, due to hosts being down for an entire data set. The third column in Tables 2 and 3 lists the percentage of lost packets that were not in runs of length 30.

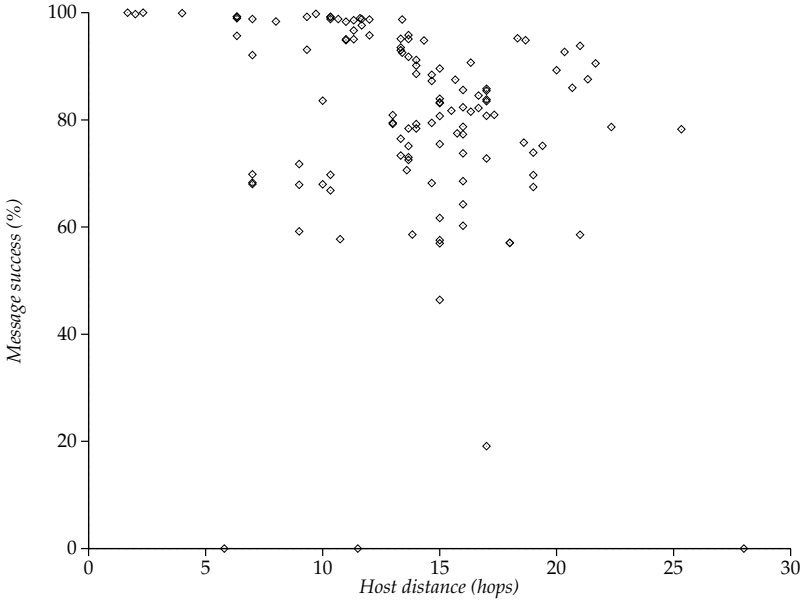

FIGURE 3: Packet delivery versus distance.

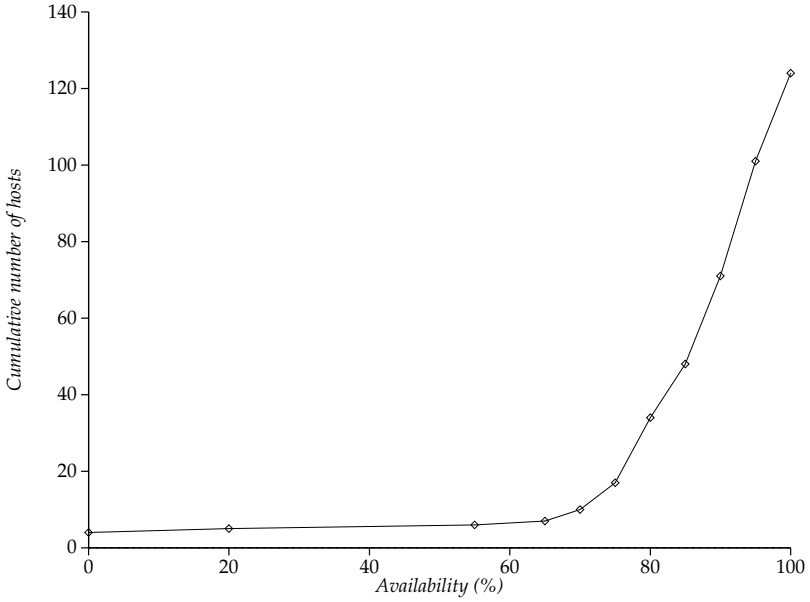

FIGURE 4: Approximate host availability. Average availability 87.5%

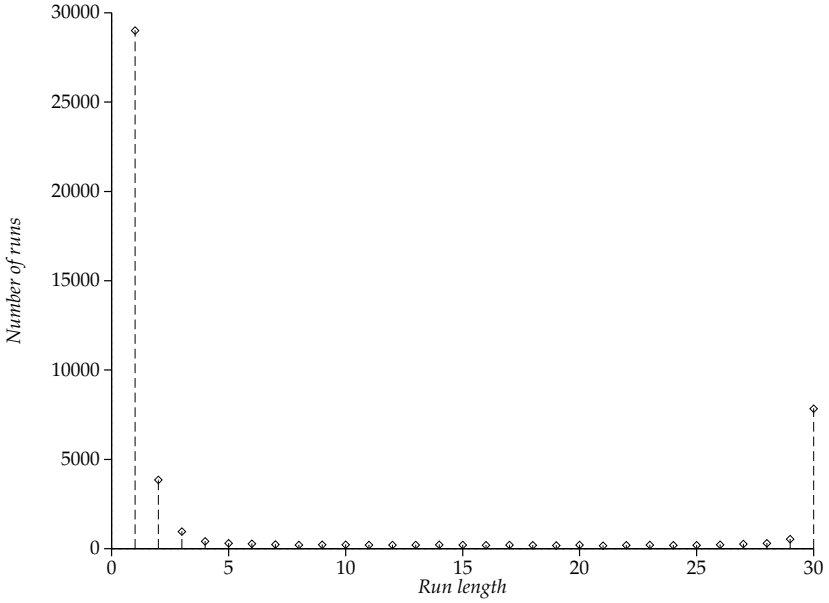

FIGURE 5: Lengths of runs of failed messages.

This number approximates the percentage of each run length that is due solely to communication failure, such as congestion, loss of connectivity, or routing loops.

I compared this distribution to what would be obtained if all communication failures were independent. This can be modeled as a Bernoulli trial with parameter f. The parameter corresponds to the probability that a packet would be successfully acknowledged. The probability $p(n)$ that a lost packet would be part of a run of length n is

$$p(n) = \frac{n(1-f)^n f}{\sum_{i=1}^{\infty} i(1-f)^i f}$$

Values of this distribution are shown in the fourth column of Tables 2 and 3, for the 24-host and 125-host experiments respectively.

If packet losses were independent, there would be many more single- or double-packet failures than were

TABLE 2: Fraction of lost packets by size of run (24-host experiment).

Length of run	Failure fraction (%) All failures	Communication	Independent $f = 93.32\%$
1	50.04	67.81	87.09
2	6.87	9.31	11.63
3	1.33	1.80	1.17
11	1.99	2.70	—
12	3.47	4.70	—
13	1.52	2.07	—
17	0.77	1.04	—
50	26.22	—	—

TABLE 3: Fraction of lost packets by size of run (125-host experiment).

Length of run	Failure fraction (%) All failures	Communication	Independent $f = 80.0\%$
1	7.73	20.63	64.00
2	1.03	2.74	25.60
3	0.26	0.68	7.68
4	0.11	0.29	2.05
5	0.08	0.21	0.51
30	2.09	—	—

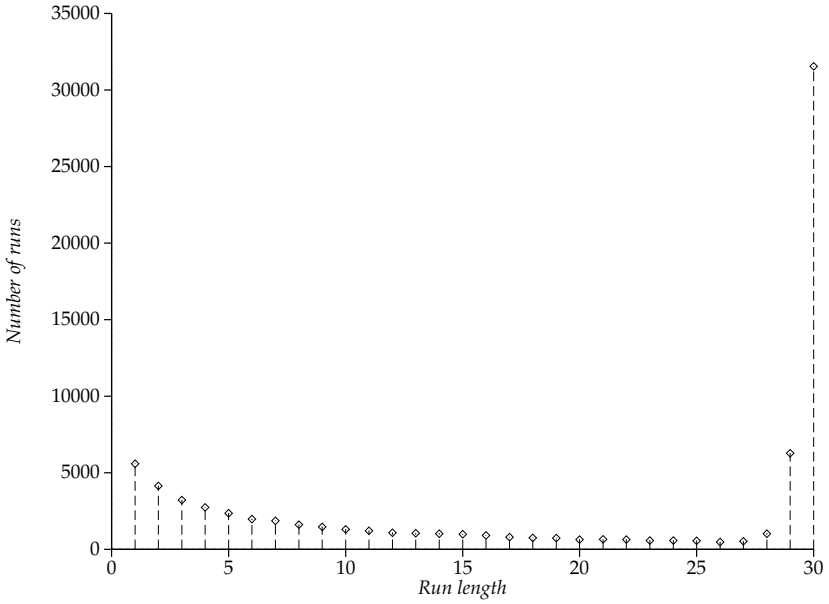

FIGURE 6: Lengths of runs of successful messages.

observed. The difference between the observed behavior and predicted behavior for independent failure leads to the unsurprising conclusion that packet losses are not independent events.

There appear to be two behaviors for message failure: short, transient failures due to temporary network conditions, and longer failures due to host or network failure. These data indicate that an internetwork communication protocol would do well to retry failed messages. Further, it appears that most of the advantage can be obtained using a small number of retries. In the 125-host traces, when a sequence of three failures has been observed there is about a 60% probability that the host will be unreachable for the entire set of 30 or 50 polls.

I also considered how many consecutive packets succeeded. As with failures, successful packets were classified by run length (Figure 6.) I found that there were many data sets in which all messages succeeded. However, I also found that there were many runs of a small number of successful messages. Short times between failures are further indication that failures tend to cluster.

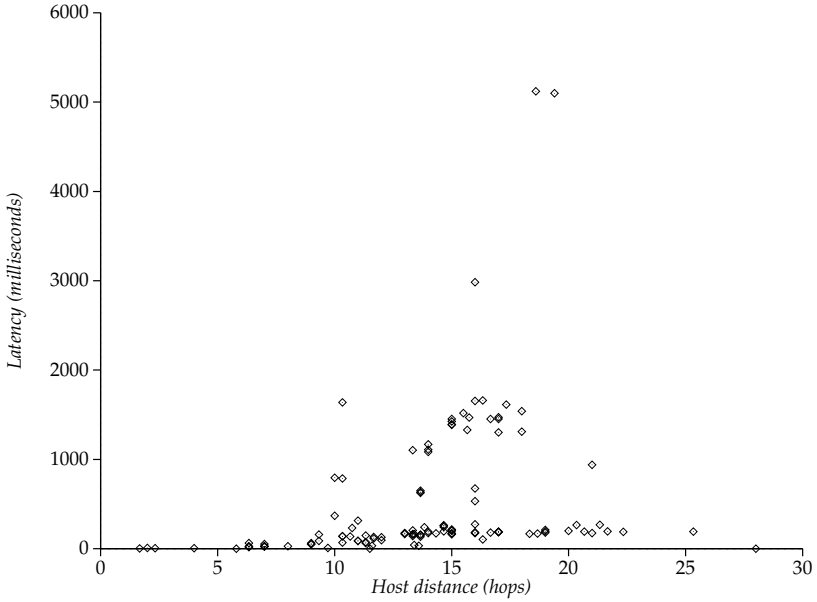

FIGURE 7: Message latency versus distance.

2.3 Packet latency distribution

Communication latency was the second focus of the measurements. The values were obtained using a Sun 4/20 workstation that has a clock resolution of approximately 10 milliseconds. (This resolution is obvious in Figures 12 and 13.) The average response latency for hosts in the first experiment is reported in Table 1.

As with packet loss, I was curious whether the number of gateways traversed in communicating with a host was related to the average latency. Figure 7 shows the latency against the distance in hops. It would appear that there may be some relation between the two.

While the average latency may be of interest, its distribution is equally important. Figures 8 through 11 present four typical distributions. These graphs show histograms of the fraction of messages that fell into 10-millisecond ranges, starting from zero. Most hosts showed a very few short-latency messages, with a sudden peak dropping rapidly back to zero.

The host **sequoia.ucsc.edu** (Figure 8) is at UC Santa Cruz, in the same organization as the host from which the measurements were taken. One gateway machine connects the Ethernets used by either machine. Most response times were sufficiently small that the 10-millisecond sampling resolution is of some concern. This curve is typical of the results observed for hosts on the same or nearby Ethernet segments. The latency distribution for **bromide.chem.utah.edu** (Figure 9) is typical of the distribution observed for hosts in North America. It is similar to that of a nearby host, but shifted toward greater latency.

The distributions for the hosts **cana.sci.kun.nl** (Figure 10), a site in the Netherlands, and **brake.ii.uib.no** (Figure 11), a site in Norway, illustrate the range of distributions observed for overseas connections. The distribution for the Dutch host appears not unlike that of a host in North America, with the majority of messages having a small latency, though the variance is quite a bit larger. The Norwegian site exhibits a much more random distribution. I believe that the packets to this host are routed through a satellite channel, which usually causes high variability.

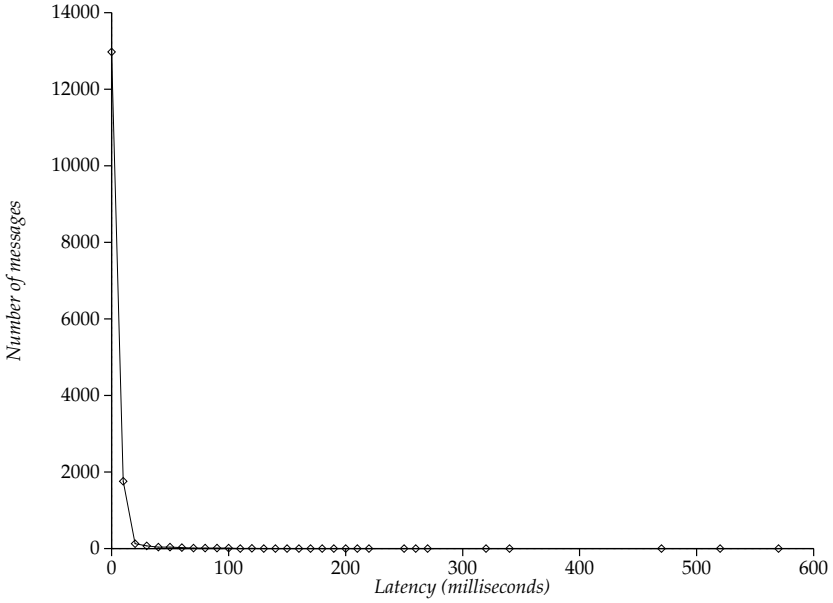

FIGURE 8: Distribution of communication latency for **sequoia.ucsc.edu**.

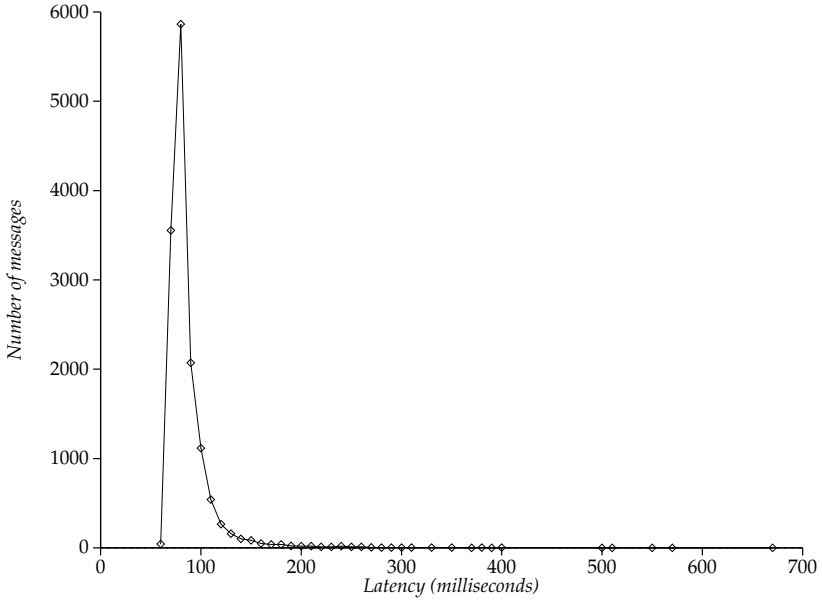

FIGURE 9: Distribution of communication latency for **bromide.chem.utah.edu**. Average latency 87 milliseconds.

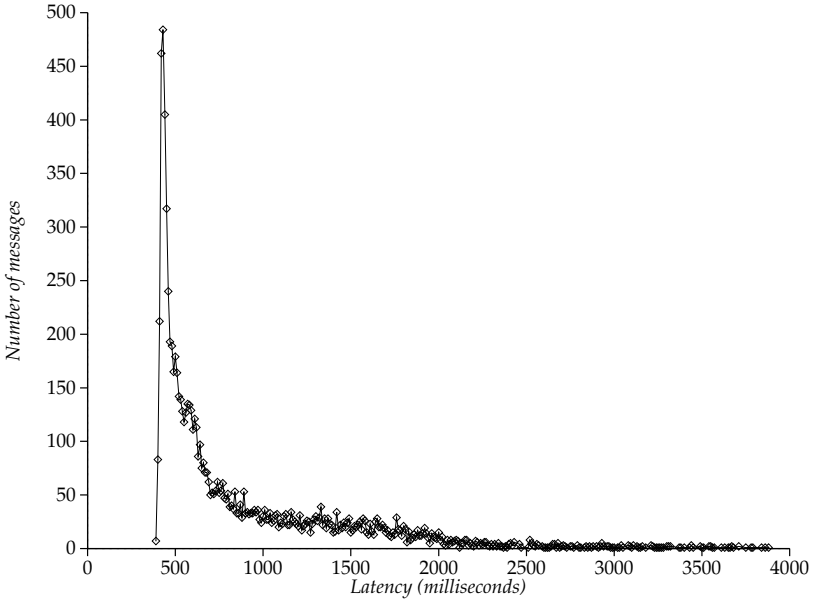

FIGURE 10: Distribution of communication latency for **cana.sci.kun.nl**. Average latency 938 milliseconds.

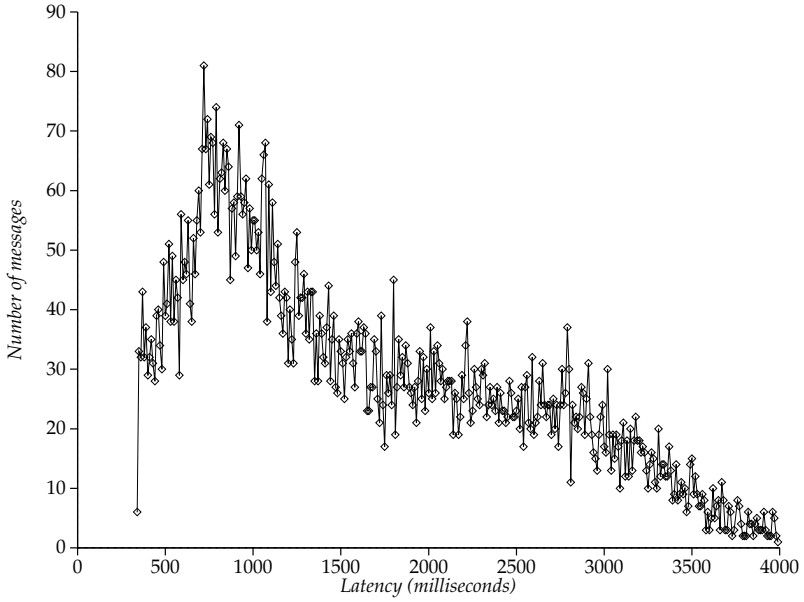

FIGURE 11: Distribution of communication latency for **brake.ii.uib.no**. Average latency 1653 milliseconds.

2.4 Predicting effective latency

The overall average values for communication latency and packet loss may not be good predictors of actual performance. It appears from the samples that failures cluster. Upon examining the traces it also appears that the latency of one message is related to the latency of the next. This section examines a way to predict latency and loss rate from recent past behavior.

There are two uses for behavior prediction. First, the prediction can be used to select probably nearby sites. Second, it can also be used to adaptively set timeout values for detecting packet loss.

The performance predictions can be based on *a priori* information, such as the topology of the network, or on observed behavior, such as past message latency. Most systems have used a *moving average* of recent behavior for prediction. This is used, for example, in most TCP implementations.

The moving average of a sequence A_i of latency samples a_i at time t is

$$\overline{a_t} = \sum_{i=o}^{t} w^{t-i} a_i,$$

where w is the weight of new samples, $0 < w \leq 1$. This can also be written as the more convenient recurrence

$$\overline{a_t} = w a_t + (1 - w)\overline{a_{t-1}}.$$

Figure 12 illustrates how the moving averages of latency behave. This figure shows 100 samples of communication latency from a trace of communication with **sequoia.ucsc.edu.** Two curves show the effects of different weighting values. As long-latency samples are observed, the moving average rises, then decays back to a lower value as latency returns to normal. Figures 13, 14, and 15 show similar curves for the other three hosts considered in the last section. The moving average can be seen to track changes in behavior. The flat sections in Figure 14 (for **cana.sci.kun.nl**) and Figure 15 (for **brake.ii.uib.no**) represent failed samples. These are ignored when calculating moving averages.

An appropriate timeout period for determining when messages have failed can be based on the predicted communication latency. The moving average at time t is an estimate of the mean of the latency distribution for the next sample. While it is obvious from the latency distributions for **cana.sci.kun.nl** (Figure 10) and **brake.ii.uib.no** (Figure 11) that message latencies are not quite exponentially distributed, they can be used as such to calculate a timeout period. The parameter λ for the exponential distribution can be estimated for a sample using $\hat{\lambda} = 1/\overline{a_t}$, the maximum likelihood estimator. The exponential is used to estimate a reasonable upper bound for message latency by taking, say, its 95th percentile. The rth percentile $a_{r,t}$ of the exponential approximating the predicted latency can be computed as

$$a_{r,t} = \frac{-\ln(1 - 0.01r)}{\hat{\lambda}} = -\ln(1 - 0.01r)\overline{a_t}.$$

For the 95th percentile, this leads to the formula

$$a_{95,t} \approx 2.995732\,\overline{a_t}.$$

This computation is different from that proposed by Jacobson for his TCP system. His approach treats latency as normally distributed, and uses an approximation of the sample standard deviation to find a likely value for the timeout. I have not yet done any systematic comparison of the two approaches, but it is worth consideration.

If this setting is to be useful for the timeout period, it must not be too short. When it is, the protocol will time out before a reply is received and either send another message or declare the host unavailable, even

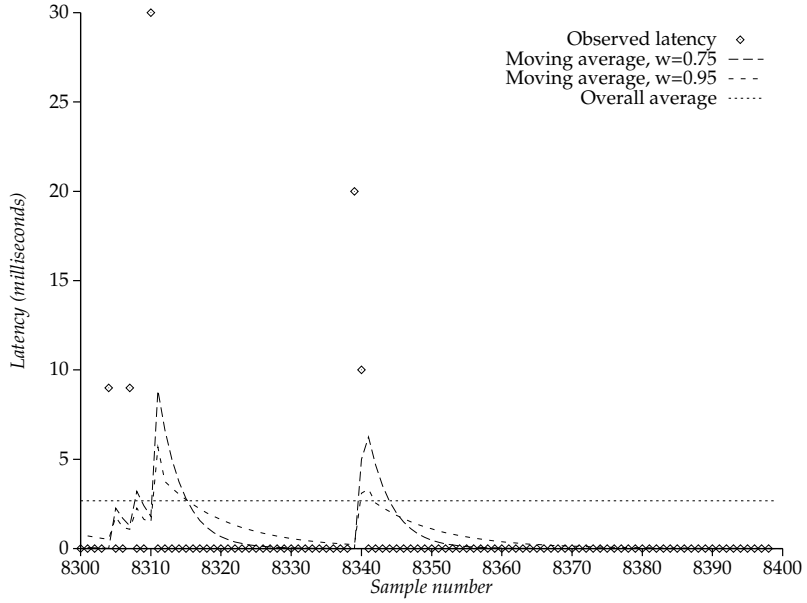

FIGURE 12: Sample moving averages of latency for **sequoia.ucsc.edu**.

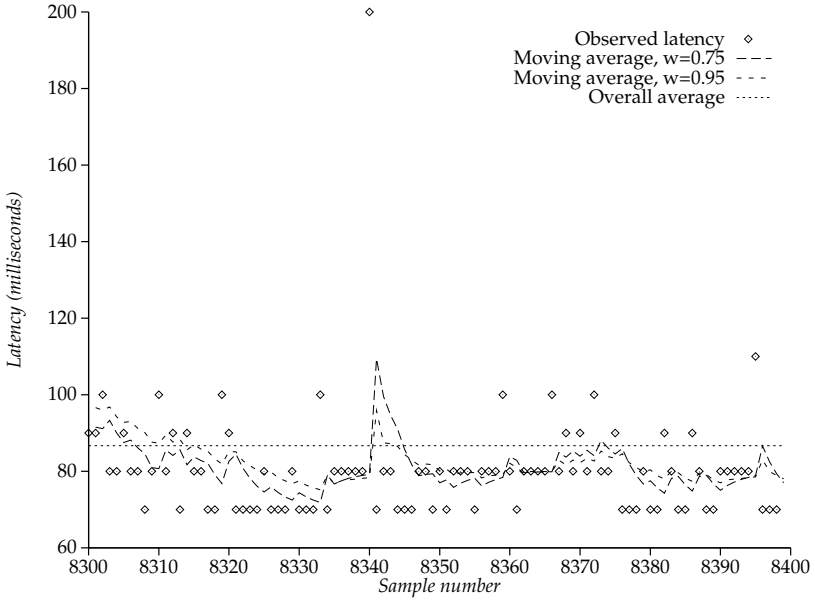

FIGURE 13: Sample moving averages of latency for **bromide.chem.utah.edu**.

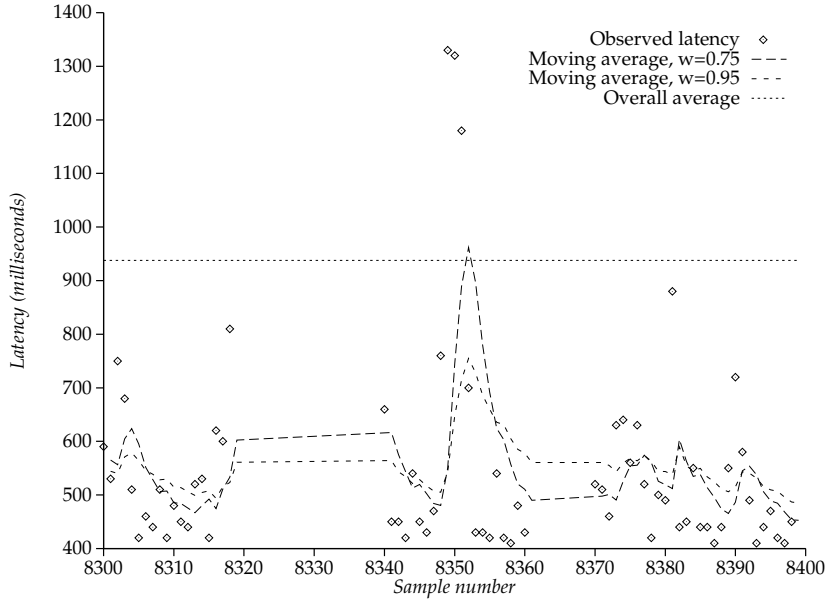

FIGURE 14: Sample moving averages of latency for **cana.sci.kun.nl**.

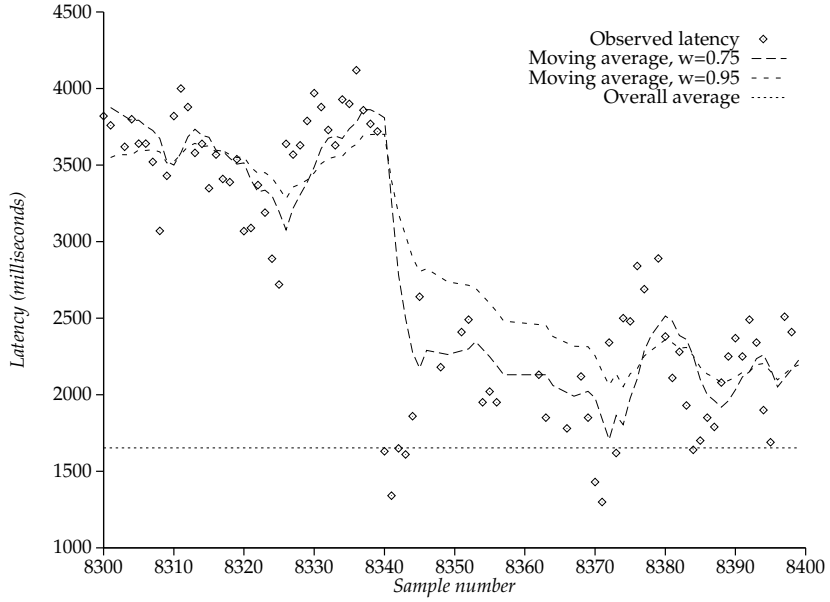

FIGURE 15: Sample moving averages of latency for **brake.ii.uib.no**.

TABLE 4: Fraction of replies rejected because of short timeout. 95th percentile used to set timeout. Moving average weight $w = 0.95$.

	Fraction rejected	
	No min	40ms min
sequoia.ucsc.edu	3.17%	1.20%
bromide.chem.utah.edu	0.27%	0.27%
cana.sci.kun.edu	0.54%	0.54%
brake.ii.uib.no	0.19%	0.19%

though the reply was on its way. The expense of retrying or declaring failure is likely to be unacceptable, so any timeout setting must not reject too many valid messages. On the other hand, the timeout period must not be too long, since a protocol must wait for that period before a message can be determined to have failed.

I examined the the traces using the 95th percentile estimator. The fraction of replies that were returned later than the estimated timeout period are shown in Table 4. It is very small for all but the nearby site, for which it was 3.17%. The estimated timeout period for this host often goes to zero. This occurs because the resolution of the trace samples is only 10 milliseconds, so the actual latency was almost always small enough to be recorded as zero. The fraction rejected dropped to 1.20% of successful replies upon applying a minimum timeout period of 40 milliseconds. While this is still high, from the distribution in Figure 8 it can be seen that the cutoff would have to be set to several hundred milliseconds to obtain less than 0.05% rejection fractions from this host. The average latency and variance are small enough that cutoff values more than about 40 milliseconds make no sense.

Moving averages can also be applied to the estimation of failure probability. Given a sequence of samples $F_i \in \{0, 1\}$, the moving average

$$\overline{f_t} = \sum_{i=0}^{t} w^{t-i} f_i$$

gives an approximation of the likelihood of failure. A large weight, that is, a value of w near one, appears to work well by accounting for short-term failure behavior.

The overall latency expectation is useful when a communication protocol is to selectively communicate with the replicas most likely to respond quickly. Given that the communication latency, probability of failure, and timeout period can all be estimated, the overall expected time o_t is the sum of the expected latency and timeout period, weighted by failure probability:

$$o_t = \overline{f_t} a_{95,t} + (1 - \overline{f_t})\overline{a_t}.$$

Figure 16 shows a sample of this overall expectation for one of the sample hosts. The latencies shown in this figure for failed messages are the timeout periods for $w = 0.95$. The way the expectation responds to changing conditions is evident in the samples between 8330 and 8350, where the expectation changes from tracking actual latency in a low-failure period to averaging actual latency and time-out period in a high-failure period.

3 Effect of using nearby sites

The quorum multicast protocols use the performance predictions discussed in the last section to attempt to optimize communication latency. These protocols provide the interface:

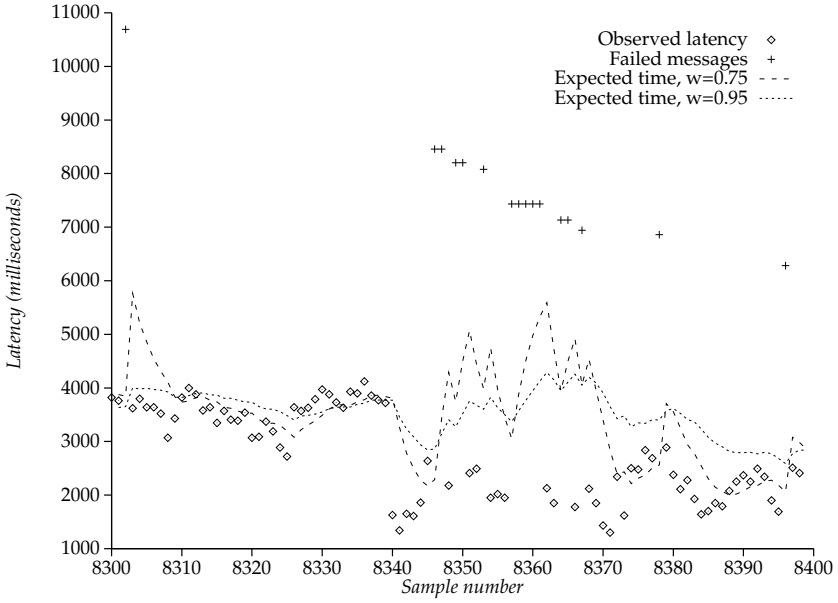

FIGURE 16: Expected communication latency for **brake.ii.uib.no**

---

quorum-multicast(message, site set, reply count, tuning parameter)
→ reply set

The message is sent to at least a **reply count** of the sites. If the protocol cannot communicate with at least that many sites, it reports an exception. The site set is assumed to be ordered (or orderable), perhaps using the latency predictions of the last section. The tuning parameter controls the eagerness with which some protocols contact less preferred sites, as will be explained shortly.

I have investigated four variations of quorum multicast protocols. The **naive** protocol is the simplest: it sends one message to each site, and waits until it either receives enough replies or times out. The **reschedule** protocol tries to only contact the best possible sites by first sending enough packets to make the reply count, and sending packets to less favored sites when the first packets time out. This time out is set to a "delay" fraction d of the normal message time out period, $0 \leq d \leq 1$. In this way the **reschedule** protocol can behave like **naive** by setting the delay d to zero, or it can use the fewest possible messages by setting d to one. The **retry** and **count** protocols improve upon **reschedule** by retrying when packets are lost. They use different policies for doing so; the details are not relevant to this paper.

Operation success is measured by the fraction of all multicast operations that were successful in meeting the reply count. In the simulations, the **naive** and **reschedule** protocols each exhibited an approximately constant success fraction, at about 82% of all operations. Since these two protocols each attempt to send at most one message to a replica, the delay parameter d has no effect on the probability of success. The **retry** protocol, which retries nearby replicas more times when the delay parameter d is larger, succeeds in more than 94% of all cases when $d \geq 0.1$, while **count** performs even better.

The data obtained by measuring a test application show that all four protocols met the reply count more than 95% of the time. **Count** succeeded more often than the other protocols for almost all values of d, with **retry** generally succeeding more often than **naive** and **reschedule**. These results are similar to the simulation results.

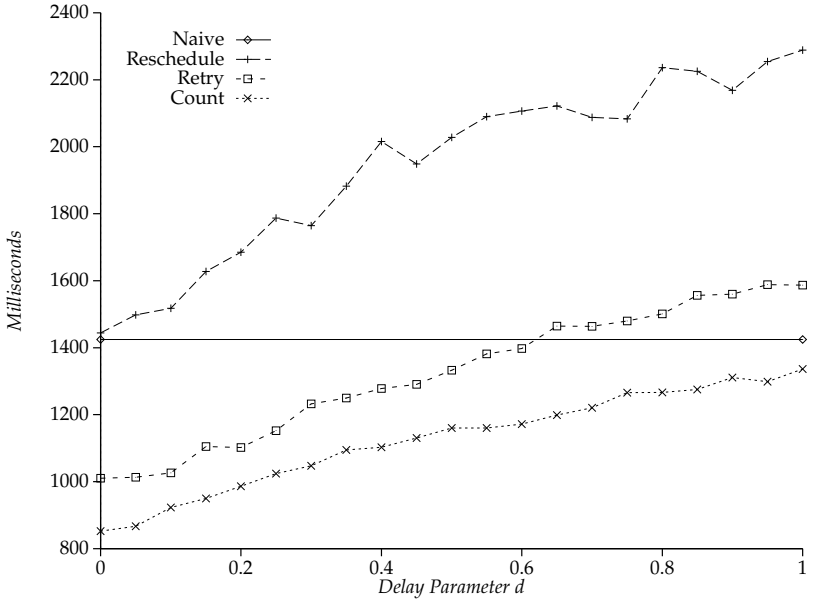

FIGURE 17: Communication latency for all operations.

For operations that are able to meet their reply count, **naive** is generally the fastest of the four protocols, since it always sends messages to every replica immediately. The communication latency for the other protocols increases approximately linearly as the delay parameter d increases, taking about the same amount of time as **naive** at $d = 0$. Of the three, **count** takes longer than **retry**, which in turn takes very slightly longer than **reschedule**. **Reschedule** takes less time than the other two because of the rare cases where the **retry** and **count** protocols must send more than one message to distant replicas to obtain the reply count.

The performance of the four protocols is quite different when the reply count cannot be met – all four protocols require several seconds to declare failure. While this is quite a long time, failures constitute only a few percent of all operations and the latency is not onerous. **Naive** is the baseline measure, requiring about 4.8 seconds to determine that a reply count cannot be obtained – almost an order of magnitude longer than was generally required for success. The latency of the other three protocols again increases roughly linearly in d. **Reschedule** requires more time than **naive** since it must detect just as many failed messages, but it may have delayed sending some of those messages. **Retry** requires more time than all the others for most values of d. **Count** performs much better than any of the other three protocols. It avoids the problem of having to communicate with the most distant replica, since it can stop when sufficient nearby replicas have failed.

The measured results differ slightly because fewer messages failed. While simulation indicated that **reschedule** takes more time than **naive** to declare failure, and that this time increases with d, the measured results show that the two have quite similar latencies. The sample size is small enough that this result is inconclusive.

Figure 17 shows the overall latency for each protocol. Since the probability of meeting the reply count is quite high, the values for successful operations predominate in these graphs. However, it is worth noting that even with a high probability of success, the low failure latency of **count** makes it the fastest of the three quorum multicast protocols, consistently faster even than **naive**. **Reschedule** has the highest latency of the three for all values of d. **Retry** is better than **naive** or **reschedule** for values of d less than about 0.6. This is the reverse of their positions for successful operations. The latency of the quorum multicast

algorithms increases approximately linearly as d increases. The overall measurement results are consistent with simulation results since the overall success rate was in excess of 95%, despite the differences in failure behavior.

4 Bandwidth

While latency and packet loss are most important for communicating with directory services, available throughput or bandwidth is most important when transferring large amounts of information from a storage server. In this section I will present the results of an experiment to measure effective bandwidth between sites, and to determine if bandwidth can be predicted.

The experiment consisted of a program that sent 1 000 000 bytes to the **discard** daemon on a remote site using TCP. The **discard** daemon, which normally listens on port 9, will receive any data and immediately discard it. Since TCP is a reliable stream protocol, the sender can estimate the bandwidth by measuring the time it takes to send the data.

This estimation will not be completely accurate. The TCP protocol dynamically adjusts its window size and timeout values, and these values might not stabilize. The measurement program also cannot directly measure the transmission time; instead, it can only measure the rate at which the local kernel or TCP implementation accepts information. It seems unlikely that the TCP implementation will be able to buffer more than a tenth of a megabyte, so the accepted bandwidth is not likely to be more than 10% greater than the actual bandwidth.

As inaccurate as this estimation of network bandwidth is, it is still arguably the right measure. An application is concerned with the actual end-to-end bandwidth, not with the maximum effective bandwidth. Factors such as kernel buffering and dynamic TCP behavior should be considered. In at least one storage system objects transferred in bulk are between one-half and two megabytes. I believe that the numbers measured here are useful.

Table **??** summarizes the experiment. The measurement program ran at four sites: **oak.ucsc.edu,** a SparcStation 2, **manray.berkeley.edu,** a SparcStation 1, **beowulf.ucsd.edu,** a SparcStation 1, and **slice.ooc.uva.nl,** a SparcStation 1. These four sites polled thirteen sites, including each other. The measurement program run for the month of March 1992, polling once every four hours. The times were adjusted so that no two sites were polling simultaneously.

The table lists the mean bandwidth observed and its variance. As expected, communication within one machine is more than an order of magnitude faster than sending packets on a network. Few of the other results were surprising: where there were low-capacity links (such as to Canada or Australia) very little bandwidth was available; sites connected close to the NSFnet backbone (Cornell and MIT) did much better. The variance, however, was much higher than expected, and dims the prospect for accurate prediction.

Figures 18 through 26 show the bandwidth distribution for several different hosts, presented in roughly increasing order of "distance." All these were measured from **beowulf.ucsd.edu,** but the results are similar to those observed from other hosts. The figures show the bandwidth for varying fractions of the samples. Accordingly, horizontal segments reflect large numbers of samples with similar bandwidth. in general, the more horizontal, the more consistent the samples.

Figure 18 shows the distribution when **beowulf** is communicating with itself. There are a number of steps in the graph, indicating that many samples showed the same bandwidth.

Figures 19 and 20 show connections from **beowulf** to hosts on the BARRnet. Both these sites show a few fast connections, as evidenced by the more vertical tail on the right, and a few slow connections, as shown by the left-hand end. The remainder of the samples are fairly uniformly distributed between 10 and 35 Kbytes/sec (for **manray**) or 20 and 50 kbytes/sec (for **oak**). The samples appear not to have an exploitable central tendency.

TABLE 5: Mean bandwidth, sampled from four sites.

	oak		manray		beowulf		slice	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
oak.ucsc.edu	1208899	184077	26778	14503	34931	14710	3425	1387
manray.berkeley.edu	18517	10943	703491	111866	25144	9565	3193	1378
beowulf.ucsd.edu	16906	9409	23520	11637	717695	237711	3209	1377
slice.ooc.uva.nl	3380	1626	3613	1939	4236	2075	636399	155168
cc.mcgill.ca	1776	2314	1699	2377	2905	3168	2237	1519
cs.wwu.edu	4874	1321	5259	1043	5459	1009	2787	1269
gvax.cs.cornell.edu	16693	5800	16022	5642	18465	4495	5791	1086
ifi.uio.no	1778	1219	1625	1120	1955	1339	5590	1039
inria.inria.fr	3050	1904	3395	2377	4186	2618	4432	1410
lcs.mit.edu	12060	6210	14410	7294	15699	8674	4446	1152
syd.dit.csiro.au	2477	809	2872	891	3042	1018	1466	652
top.cs.vu.nl	4844	2518	4693	2863	5745	2959	6609	302
uhunix.uhcc.hawaii.edu	13362	6800	20641	7165	18730	5585	3522	1289

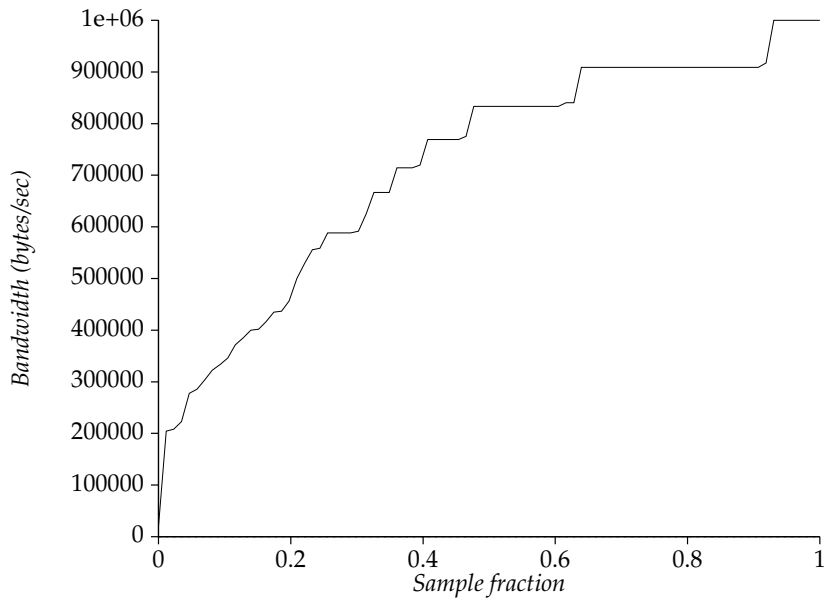

FIGURE 18: Bandwidth distribution from **beowulf** to itself.

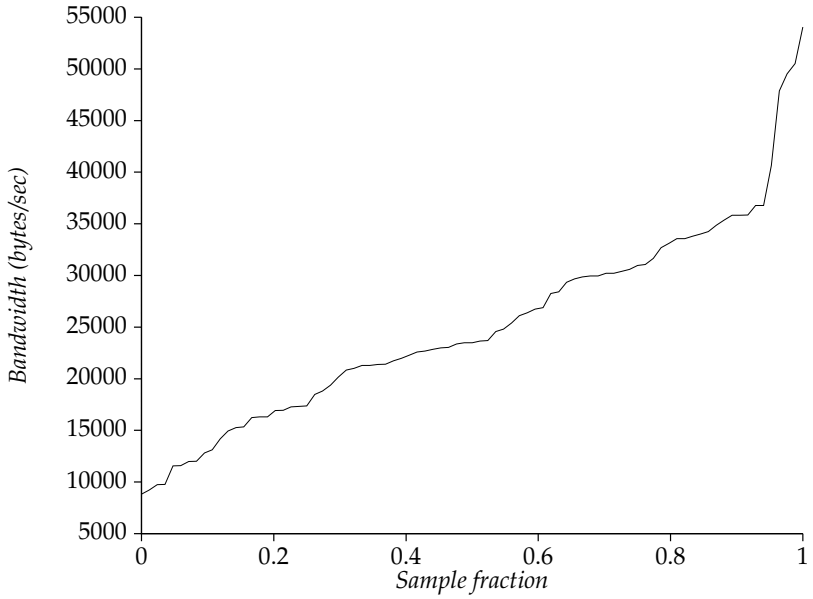

FIGURE 19: Bandwidth distribution from **beowulf** to **manray**.

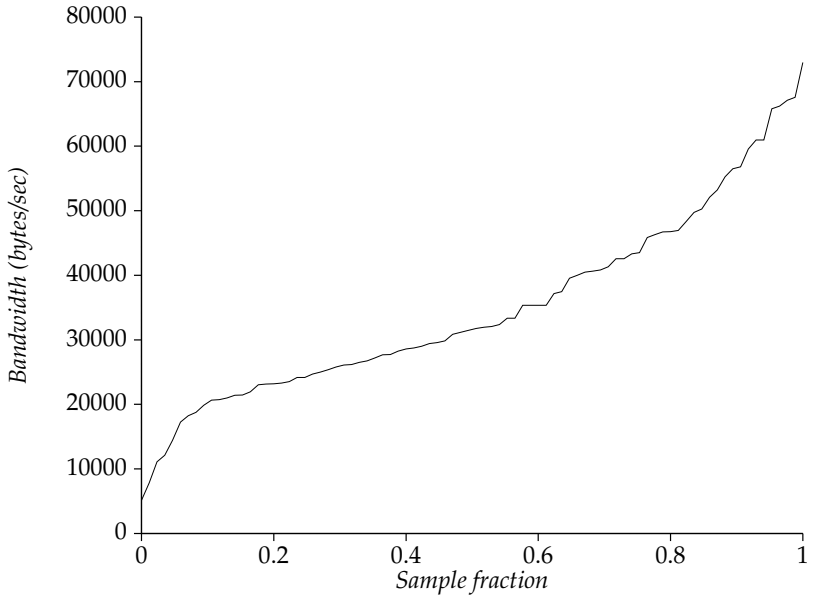

FIGURE 20: Bandwidth distribution from **beowulf** to **oak**.

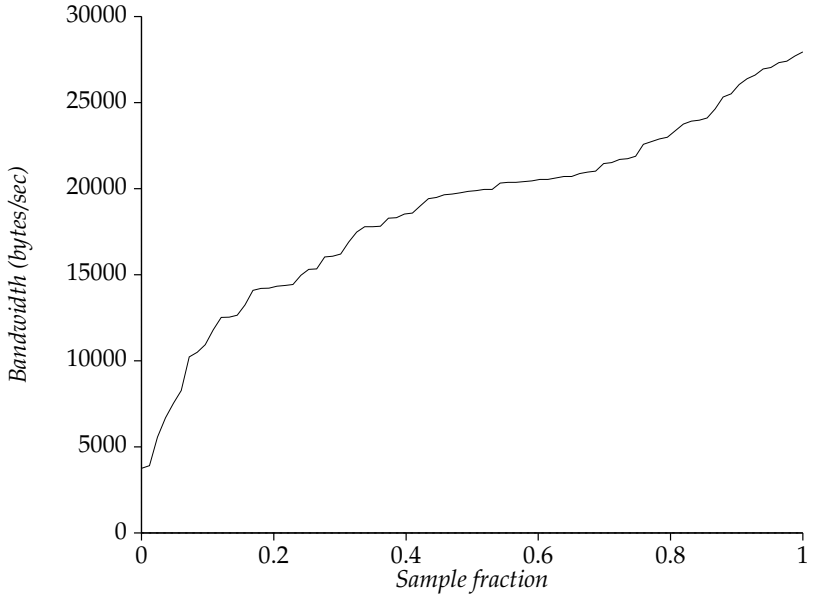

FIGURE 21: Bandwidth distribution from **beowulf** to **hawaii**.

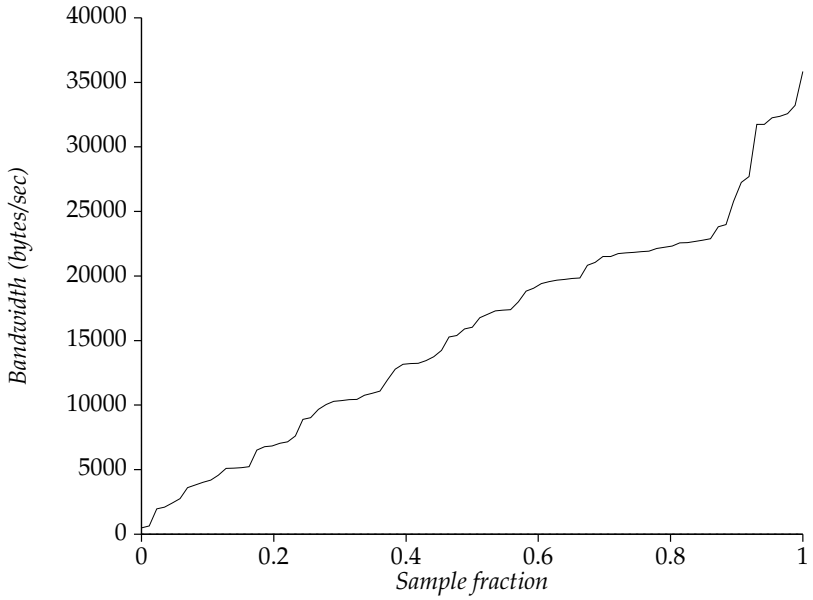

FIGURE 22: Bandwidth distribution from **beowulf** to **mit**.

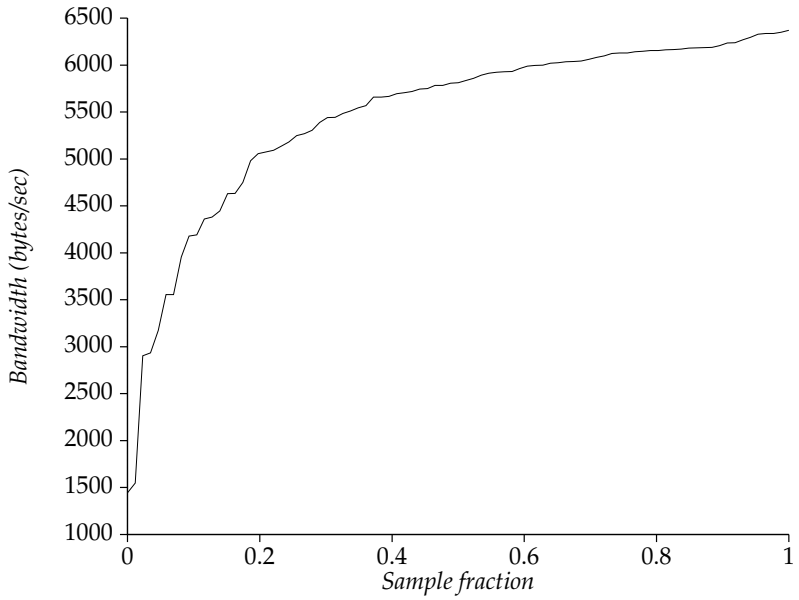

FIGURE 23: Bandwidth distribution from **beowulf** to **wwu**.

Figures 21, 22, and 23 show samples taken to other US sites. The site in Hawaii is connected through NASA Ames, and exhibits the best performance of the three. The site at MIT is near an NSFnet gateway, but shows almost an almost uniform bandwidth distribution with a low near zero, excepting a slight spike at around 20 kbytes/sec. The site in Washington is connected using a low-speed modem to NorthWestNet, and provides bandwidth between 5 and 6 kbytes/sec about 80% of the time.

The final three figures show connections outside the US. Figure 24 shows samples for McGill University, which uses a nearly-saturated 56 kbit/sec link to the US. The saturation is evident from graph: most of the samples show little bandwidth is available, but there is the occasional sample many times larger. This link is heavily used by the **archie** resource-location service. Figure 25 shows traffic to the Netherlands, and Figure 26 shows traffic to Australia. Both of these use low-speed transoceanic cables; performance is much lower than for most North American sites. However, the links do not appear to be saturated.

4.1 Predicting bandwidth

Bandwidth prediction is primarily useful on the Internet for ranking sites. One site will often have a choice of several others from which it can transfer some information, and it should be able to determine which site will provide the fastest service.

This kind of prediction is not generally useful for multimedia applications, which need guaranteed performance. The Internet as currently built does not provide these guarantees, and the kind of performance prediction discussed in this paper does not provide a mechanism for them.

As with latency, I have used moving averages of recent measurements as a predictor. I performed two variations on the analysis: a “cold start” evaluation that started all estimates at zero, and a “warm start” evaluation that first found the mean of all samples, then used that as an initial estimate. The former analysis should show how accurate the predictor is for the first several samples; the latter should show how the system behaves once it has reached a steady state.

Tables 6 and 7 show how accurate this is. For these measures, I computed the absolute error between

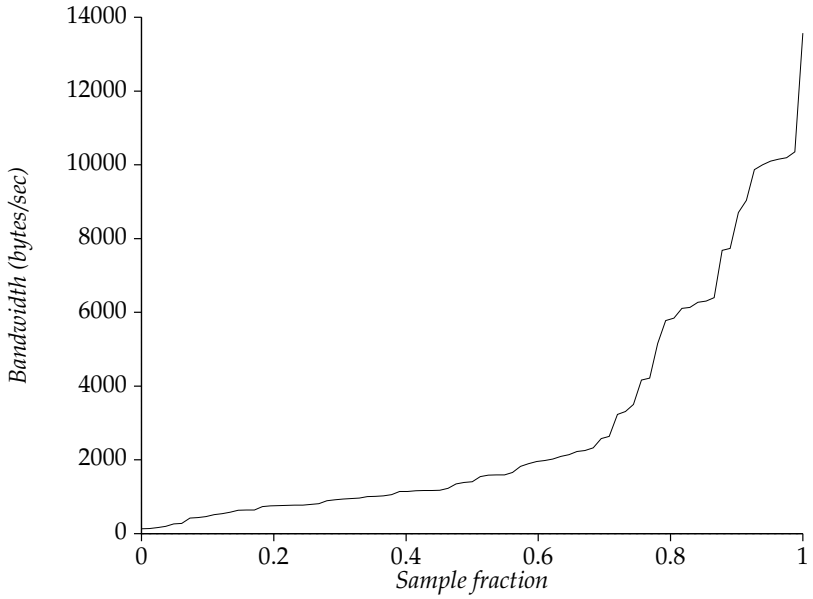

FIGURE 24: Bandwidth distribution from **beowulf** to **mcgill**.

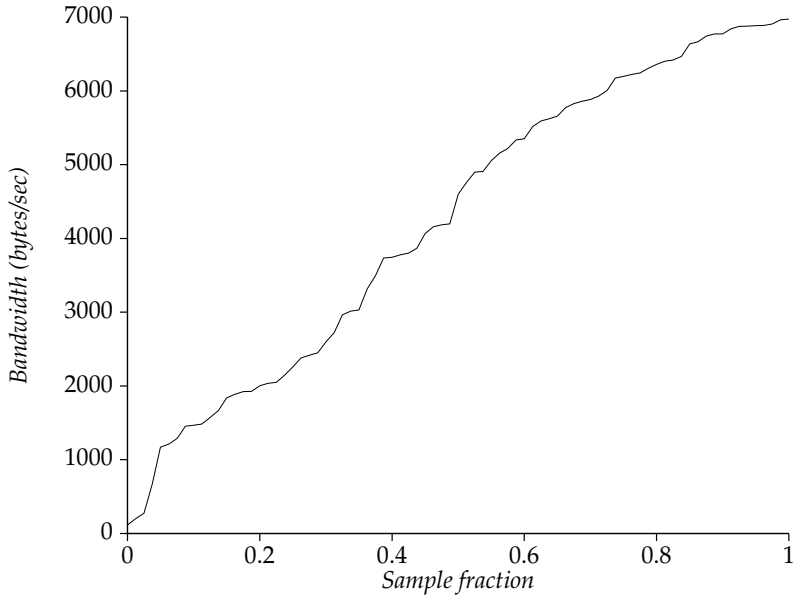

FIGURE 25: Bandwidth distribution from **beowulf** to **slice**.

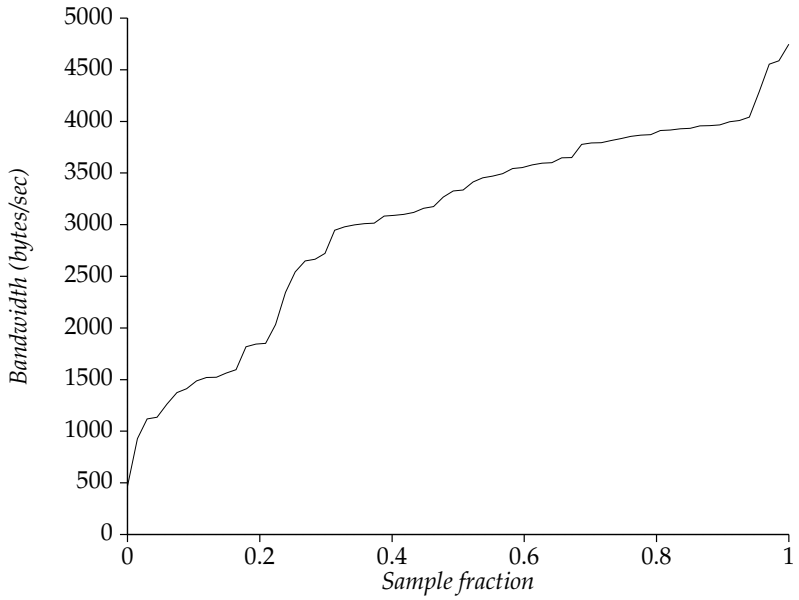

FIGURE 26: Bandwidth distribution from **beowulf** to **csiro** (Australia).

each sample and the moving-average prediction, and report the mean error and its deviation. The moving average used a weight of $\alpha = 0.9$, so it responds slowly to changes.

It appears that the moving average is not a particularly good predictor. The mean error for most sites is between 30 and 50% of the mean bandwidth, and the variance is high. The warm-start error is generally lower than the cold-start error, but the difference is not necessarily statistically significant. This indicates that either the moving average converges quickly, or the variability of bandwidth measurements is such that this predictor will not work well. I suspect the latter from the sample variance. A conditional autocorrelation of bandwidth samples should give some insight into the usability of this sort of predictor, by showing how much short-term and long-term variation is in the samples.

A site will often need only to be able to rank sites, rather than determine accurately the real available bandwidth. This is a somewhat simpler problem, since it may be less sensitive to short-term performance variations.

Table 8 shows how well moving-average predictions can be used to order sites. The table reports how much lower observed bandwidth is than the optimal, when moving averages are used to rank one, two, or three sites from the four measured. There is fairly high variance in this difference. The observed bandwidth from one site to another single best site is, on the average, less than ideal by a few kilobytes per second; however, this value is often 20–30% of the mean bandwidth measured from that site. This indicates that prediction is not working especially well, and that it behaves very badly at times. This is not surprising, given the variance in actual measured bandwidth.

4.2 Static prediction

The dynamic prediction mechanism discussed in the last section only responds to changes when communication has been attempted. It can only work well when communication is frequent. If a site is ranked as undesirable, or if a new site becomes available, there may be no up-to-date prediction information available. To remedy this, a site must be polled proactively to obtain the needed information.

TABLE 6: Mean error in "cold start" bandwidth prediction, in bytes per second, using a moving average with parameter $\alpha = 0.9$ and an initial value of zero.

	oak		manray		beowulf		slice	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
oak.ucsc.edu	233701	266734	11923	10407	12941	11505	1263	935
manray.berkeley.edu	7614	8967	135152	162386	8863	7578	1235	946
beowulf.ucsd.edu	7253	7439	9812	8618	220178	178106	1155	909
slice.ooc.uva.nl	1418	1038	1723	1047	1864	1199	160777	156185
cc.mcgill.ca	1511	1632	1482	1719	2458	2199	1311	828
cs.wwu.edu	1279	1122	1228	1169	1104	1113	1109	767
gvax.cs.cornell.edu	5505	4497	5604	4137	4348	3726	1244	1304
ifi.uio.no	953	873	846	834	1009	988	1258	1236
inria.inria.fr	1574	1295	2027	1397	2430	1497	1406	1151
lcs.mit.edu	5032	4298	6277	4021	7894	5855	1193	1013
syd.dit.csiro.au	825	636	970	737	1068	753	578	503
top.cs.vu.nl	2194	1726	2467	1711	2727	1486	899	1406
uhunix.uhcc.hawaii.edu	5877	4841	6431	5599	5213	4062	1170	880

TABLE 7: Mean error in "warm start" bandwidth prediction, in bytes per second, using a moving average with parameter $\alpha = 0.9$ and an initial value of $\overline{x}$.

	oak		manray		beowulf		slice	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
oak.ucsc.edu	123840	142136	11143	9906	11691	8229	1083	766
manray.berkeley.edu	7282	8510	65539	95656	7838	6017	1092	744
beowulf.ucsd.edu	6940	6323	9472	6911	195946	138013	1060	773
slice.ooc.uva.nl	1287	855	1607	947	1754	967	108590	119251
cc.mcgill.ca	1566	1565	1545	1661	2549	2018	1276	708
cs.wwu.edu	994	867	732	767	736	705	956	686
gvax.cs.cornell.edu	4589	3251	4507	3071	3469	2497	793	793
ifi.uio.no	925	750	865	675	997	855	799	703
inria.inria.fr	1473	1102	1915	1246	2232	1331	1222	745
lcs.mit.edu	4504	3694	5480	3528	7042	4362	911	677
syd.dit.csiro.au	596	465	743	515	856	583	526	385
top.cs.vu.nl	2097	1319	2256	1499	2503	1366	234	207
uhunix.uhcc.hawaii.edu	5288	4701	5938	4021	4483	3152	969	701

TABLE 8: Loss of aggregate bandwidth from one, two, and three sites selected using moving average predictions.

	One site		Two sites		Three sites	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
oak.ucsc.edu	0	0	3637	7563	0	0
manray.berkeley.edu	0	0	2444	6031	0	0
beowulf.ucsd.edu	163	1475	2257	5763	0	0
slice.ooc.uva.nl	0	0	1128	1526	1203	1460
cc.mcgill.ca	1960	2562	1740	2801	1256	2050
cs.wwu.edu	460	811	536	989	158	457
gvax.cs.cornell.edu	3047	3895	3029	3848	79	440
ifi.uio.no	22	129	575	778	497	708
inria.inria.fr	1429	1835	1591	1967	1151	1342
lcs.mit.edu	3745	5187	3782	5023	330	864
syd.dit.csiro.au	521	784	549	737	149	395
top.cs.vu.nl	1394	1489	1707	1905	1550	1727
uhunix.uhcc.hawaii.edu	3557	4775	2244	4958	0	0

One solution is to pool measurements from everywhere on the Internet to generate global, static prediction information. Global information has the advantage of being centrally computable, and not requiring large amounts of proactive sampling. It can also be adjusted according to network-wide policies to, say, favor communication between sites in the same country.

I have investigated one such measure for predicting bandwidth between two sites, a and b. If the Internet consisted of an infinite-bandwidth backbone to which all sites were connected, then the bandwidth between any two sites would be the lesser of the bandwidths of the two connections to the backbone. Except for the site with the fastest connection, the bandwidth of a backbone connection would be equal to the maximum bandwidth observed between that site and any other site. If the bandwidth observed between two sites is denoted $B(a,b)$, the estimated bandwidth is

$$\hat{B}(a,b) = \min[\max_{c \neq a,b}(B(a,c)), \max_{c \neq a,b}(B(b,c))].$$

Table 9 shows the estimated bandwidth calculated for each of the four polling sites. This is always a poor estimator for communicating within a machine, since sending packets on a network always takes longer than simply buffering them internally. Between different machines, however, the estimator seems to reflect reality. These estimations can be averaged to give a single global measure. This is shown in the last row of Table 9.

The estimator is to be used to rank sites according to predicted available bandwidth. If this is a good estimator, it should always select the site with the greatest available bandwidth. Different estimation methods can be compared by computing how close they come to the best possible selection.

Table 10 shows this comparison. Each of the thirteen sites measured are assumed to be repeatedly attempting to communicate with the four polling sites, and must select the best one, two, or three sites. The table compares the global static measure just described with a dynamic moving-average estimator and with random selection. The bandwidth obtained using each of these estimators is compared to the optimal, and the table shows the overall difference.

A dynamic moving-average selection appears to be the best selection method. It almost always gives a better selection than static selection, and when it does not the difference is minimal. Dynamic selection also properly handles cases where a host can communicate with itself, which the static measure does not.

TABLE 9: Transitive bandwidth estimations $\hat{B}(a,b)$, and the global static measures derived from it.

	oak		manray		beowulf		slice	
	Predict	Actual	Predict	Actual	Predict	Actual	Predict	Actual
oak	18517	1208899	16906	26778	18517	34931	5791	3425
manray	16906	18517	26778	703491	26778	25144	5791	3193
beowulf	18517	16906	26778	23520	34931	717695	5791	3209
slice	5791	3380	5791	3613	5791	4236	6609	636399
Metric	13738		16492		17029		5791	

TABLE 10: Comparison of the aggregate bandwidth loss from one, two, and three sites selected using either dynamic (moving-average) or static estimations.

	One site			Two sites			Three sites		
	Stat	Dyn	Rand	Stat	Dyn	Rand	Stat	Dyn	Rand
oak.ucsc.edu	1175960	0	892056	1188259	3637	611790	0	0	315587
manray.berkeley.edu	674748	0	513407	2444	2444	354510	0	0	183415
beowulf.ucsd.edu	163	163	531820	2257	2257	365556	0	0	188656
slice.ooc.uva.nl	637264	0	478300	639073	1128	320508	639397	1203	161097
cc.mcgill.ca	1713	1960	2455	2105	1740	2401	1628	1256	1536
cs.wwu.edu	427	460	1288	516	536	2026	158	158	1970
gvax.cs.cornell.edu	3227	3047	7377	4030	3029	10033	79	79	8602
ifi.uio.no	3673	22	2904	4514	575	2606	4485	497	1602
inria.inria.fr	1658	1429	2072	2553	1591	2576	2458	1151	1770
lcs.mit.edu	4161	3745	8276	3392	3782	10235	330	330	7550
syd.dit.csiro.au	447	521	1029	480	549	1462	149	149	1143
top.cs.vu.nl	2072	1394	2386	3386	1707	2954	3269	1550	2172
uhunix.uhcc.hawaii.edu	5398	3557	10049	2244	2244	13518	0	0	10556

Both these methods are noticeably better than a random selection of a single site, but random selection gets better as the number of sites selected increases. When three of the four sites are to be selected, random is often better than the static selection.

In general, it appears that the global static estimator does not work as well as a local dynamic estimator.

5 Future directions

These measurements represent only a preliminary look at end-to-end performance prediction. There are several improvements to be made to these analyses.

All of the measurements can be improved by increasing the amount of data. The latency measurements only cover one week for a select few hosts; a longitudinal study, covering more hosts, will allow a real analysis of the long-term stability of predictions and of the way the predictions respond to changes in the network.

The bandwidth measurements likewise suffer from lack of data, and could be improved by obtaining more samples, between more hosts, and at finer resolution. Diurnal effects are currently invisible. These measurements could also collect routing information so that possible relations between bandwidth and number of hops could be evaluated. A conditional autocorrelation may show interesting relationships between one measurement and future measures for bandwidth, errors, and latency.

It is likely that performance prediction need not be done on an individual host-to-host basis. Instead, a body of predictions can probably be shared among all the hosts in an organization. It seems likely that most hosts will be on one or a few LANs and will share a few common routes to the rest of the Internet. One would expect that all the hosts would therefore show similar communication behavior. Sharing the prediction information should improve the quality of predictions, by making more data available, and should reduce the frequency with which a prediction system should need to proactively collect performance information.

At present it appears that static quality-of-service measures do not work well. This is unfortunate, since it means sites can only use information they have collected, and may have to proactively collect to determine what sites are distant. These effects can be mitigated somewhat by sharing a prediction database within an organization.

Acknowledgments

This work was motivated by discussions with Alan Emtage and Peter Deutsch of McGill University. Matthew Lewis (Universiteit van Amsterdam), Eric Allman (Mammoth Project, UC Berkeley), Fred Korz (Columbia University), and Darrell Long (UC Santa Cruz) helped provide accounts for running the measurements. George Neville-Neil provided encouragement, proofreading, and helpful discussion.

References

[Emtage92] Alan Emtage and Peter Deutsch. archie – an electronic directory service for the Internet. *Proceedings of Winter 1992 Usenix Conference* (San Francisco, 24–24 January 1992), pages 93–110 (January 1992).

[Golding91] Richard A. Golding. Accessing replicated data in a large-scale distributed systems. Master's thesis; published as Technical report UCSC–CRL–91–18 (June 1991). Computer and Information Sciences Board, University of California at Santa Cruz.

[Golding92] Richard A. Golding and Darrell D. E. Long. Quorum-oriented multicast protocols for data replication. *Proceedings of 8th International Conference on Data Engineering* (Tempe, Arizona, February 1992), pages 490–7 (February 1992). IEEE Computer Society Press.

[Jacobson88] Van Jacobson. Congestion avoidance and control. *Proceedings of SIGCOMM '88*, pages 314–29 (1988).