A Network Performance Tool for Grid Environments

Craig A. Lee
James Stepanek
Computer Science and Technology
The Aerospace Corporation
{lee,stepanek}@aero.org

Rich Wolski
Department of Computer Science
University of Tennessee, Knoxville
rich@cs.utk.edu

Carl Kesselman
Information Sciences Institute
University of Southern California
carl@isi.edu

Ian Foster
Mathematical and Computer Sciences
Argonne National Laboratory
foster@mcs.anl.gov

Abstract

In grid computing environments, network bandwidth discovery and allocation is a serious issue. Before their applications are running, grid users will need to choose hosts based on available bandwidth. Running applications may need to adapt to a changing set of hosts. Hence, a tool is needed for monitoring network performance that is integral to the grid environment. To address this need, *Gloperf* was developed as part of the Globus grid computing toolkit. Gloperf is designed for ease of deployment and makes simple, end-to-end TCP measurements requiring no special host permissions. Scalability is addressed by a hierarchy of measurements based on group membership and by limiting overhead to a small, acceptable, fixed percentage of the available bandwidth. Since this fixed overhead may push host-pair revisit time into the tens-of-hours, we also quantitatively examine the "trajectory" of the cost-error trade-off for measurement frequency.

1 Introduction

In grid computing environments, network bandwidth discovery and allocation is a serious issue. When starting a grid application, how do users know which hosts have suitable bandwidth among them? In the absence of any type of bandwidth reservation mechanisms, how should applications initially configure and subsequently adapt themselves to prevailing network conditions? While running applications can monitor their own performance, they cannot readily measure performance in other parts of the environment. In light of these considerations, a tool for monitoring network performance in grid computing environments is needed that is part of the grid infrastructure itself. However, performance monitoring is not enough if the information is to be used to make resource allocation decisions. Instead, a scheduler (be it a human being or an automatic scheduling program) must use the performance data that has been collected to *predict* what bandwidth will be available when an application will consume it [1]. Gloperf bandwidth information can be used to make these predictions.

To address these needs, *Gloperf* was developed as part of the Globus grid computing toolkit [15] with consideration of all fundamental issues:

- *Accuracy vs. Intrusiveness.*
- *Scalability.*
- *Portability.*
- *Fault Tolerance.*
- *Security.*
- *Measurement Policy.*
- *Data Discovery, Access and Useability.*

Gloperf is designed for ease of deployment to enhance portability. It makes simple, end-to-end TCP measurements such that no special host permissions or specific knowledge of link topology is required. Besides measuring what applications are more likely to experience, this also enhances Gloperf's fault-tolerance. Data discovery and access is provided by storing Gloperf data in a directory service that uses a well-known naming schema.

Scalability of such a tool is extremely important since grid environments have already been built that encompass thousands of nodes. $O(n^2)$ measurements of all pairs can be avoided by using a hierarchy of host-pair measurements [17]. Monitoring overhead can also be limited simply by measuring less frequently.

When designing and deploying Gloperf, however, we realized that even with hierarchical measurement, the host-pair revisit time may in the tens-of-hours if the total overhead is limited to a small, acceptable, fixed percentage of the available bandwidth. Typically performance prediction error is minimized by attempting to track highly volatile network performance with frequent measurements. Rather than doing this, we endeavored to expose the magnitude of the *prediction error* when the measurement period exceeds twelve hours (which is the anticipate measurement frequency for Gloperf when fully deployed). Hence, besides presenting the design of Gloperf, we also quantitatively examine the "trajectory" of the cost-error trade-off such that users and site administrators can intelligently choose the monitoring overhead that best meets their accuracy requirements while not exceeding their pain threshold.

2 Related Work

Network performance measurement is an extremely broad area and we can only briefly mention some of the more relevant work. Carter and Crovella [3, 2] present *bprobe* and *cprobe* to measure the bottleneck link speed and competing traffic, respectively, on a path using ICMP ECHO packets. Since these tools do not use TCP, they do not capture any TCP-related effects that an application might experience. Van Jacobson's *pathchar* [7] estimates bandwidth on all hops of a path and hence can take a very long time. It also requires root access, making it less desirable for grid environments. *TReno* [10] emulates an idealized TCP which makes the measurements independent of host-specific TCP implementations but not representative of what applications would experience. *TReno* also needs root access. *topology-d* [11] uses *ping* and

netperf to make measurements between all pairs within a *group* and then computes a minimum-cost *logical topology*. The Network Weather Service (NWS) [17] uses TCP to send small, fixed-size probes measuring in the kilobytes with a frequency that is tunable, but typically ranges from tens of seconds to several minutes. Performance measurement systems, such as the National Internet Measurement Infrastructure (NIMI) project [12], however, are being designed for arbitrary Internet hosts. This work is complemented by the Cooperative Association for Internet Data Analysis (CAIDA) [5] whose goal is to develop metrics and tools for analyzing traffic across the Internet.

Finally we note that the data from such tools are intended to be used by "higher-level" systems. Remos [9] also provides an API whereby applications can pose *flow* and *topology-based queries*. AppLeS (Application Level Scheduler) [14] uses information from the NWS to schedule distributed, resource-intensive applications. Ultimately other Globus components will use Gloperf information in similar resource discovery and allocation functions.

3 The Gloperf Design

Any performance monitoring system can be broken into four broad functional areas: (1) sensors, (2) collation of data, (3) production of derived data, and (4) access and use of data. Gloperf is implemented as a set of *gloperfd* daemons that act as the sensors. Gloperf data is collated into the Globus Metacomputing Directory Service (MDS) [4] which is a Lightweight Directory Access Protocol (LDAP) server [6]. The MDS implements a structured information naming schema, thus allowing any MDS client to access Gloperf data. Gloperf also relies on the MDS for information that allows each *gloperfd* to follow the measurement policy. Each daemon does *peer* and *group discovery* based on MDS information thus determining which peers to make measurements with.

When Globus is started on a host, one *gloperfd* will be started that registers itself by writing an object into the MDS. After registration, *gloperfd* goes into its main control loop that has the following psuedo-code behavior:

```
while ( no termination signal )  {
  query MDS for all gloperfds;
  filter list of gloperfds on version number and groups;
  build a randomized list of tests that does a
      bandwidth test and a latency test to each filtered peer;
  for each element of the list  {
    perform the test;
    write the results in the MDS;
    wait n minutes;
  }
}
```

gloperfd loops until it is told to terminate by an external signal. On each iteration, *gloperfd* queries the MDS for all other *gloperfd*s to do peer and group discovery. Group membership is denoted by the group names appearing in the daemon's MDS object. *gloperfd* only does tests to peers that are in the same groups.

Groups are currently administered by hand. Daemons are in a group "local" to their site by default. Site administrators must promote a local host to also be a member of the "global" group. This simple group mechanism allows an arbitrary hierarchy of measurements to be specified. Once a daemon has identified its peers, it constructs a randomized list of bandwidth and latency tests to each peer.

Gloperf does end-to-end bandwidth and latency measurements based a "librarized" version of *netperf* [8]. Gloperf uses a netperf "TCP_STREAM" test such that TCP/IP overhead is included in the measured throughput. Send and receive buffer sizes are the local system defaults. Hence, the performance observed is what an untuned application would observe. Gloperf uses a netperf "TCP_RR" test to measure latency as the inverse of "transactions per second".

One bandwidth or latency test in the randomized list is done every n minutes. Currently $n = 5$ minutes by default. When both bandwidth and latency tests have been done for a peer, the data is stored with the source host object in the MDS. Note that these tests are unidirectional; different tests are done from both ends of a host pair and written under the source host object.

Once the list of tests has been serviced and no termination signal has been caught, *gloperfd* queries the MDS again since other daemons may have been added to or deleted from the testbed.

Design Discussion.

Gloperf does simple, active, end-to-end measurements that require no special host permissions or site-specific knowledge. It is a decentralized design since each daemon does its own "autoconfiguration" of peer/group discovery and scheduling/reporting of test results. While the Gloperf daemons rely on the MDS, there is no interaction between the daemons other than that required for individual tests, i.e., there is no distributed scheduling of measurements. This simplified, decentralized design enhances Gloperf's fault tolerance since a failed host or link only affects the tests that require those resources. We note that all Gloperf daemons and hosts are monitored by the Globus Heartbeat Monitor (HBM) [13] which can be used to build arbitrary fault-recovery mechanisms.

Scalability and *intrusiveness* are major issues for tools such as Gloperf. Scalability is greatly enhanced by using a measurement hierarchy based on group membership. This reduces the total number of tests and the number of "redundant" tests over shared paths such as the backbones between institutional sites. This does, however, mean that some end-to-end performance has to be estimated from a "path" of end-to-end measurements. While this complicates use of the data, we can at least control the length of the paths via the number of groups.

Overall intrusiveness can be bounded simply by limiting each *gloperfd* to do no more than one test every k minutes. Hence, a five-second test every five minutes limits the overhead to 1.6% on a time-spent-testing basis. This policy has the undesirable property, however, of monotonically increasing the revisit time between any given host pair as the total number of hosts increases. While this effect is ameliorated by the group hierarchy, there are still $O(n^2)$ tests between n local host pairs in a local group. A site with ten hosts, for example, could see a 90-minute revisit time for a given local host pair. Given that we do not want to complicate the deployment and configuration of Gloperf and we want to minimize Gloperf's intrusiveness, how infrequently can we measure and still get useful data?

4 Performance Results

Ideally what is desired are highly accurate measurements that are minimally intrusive. Unfortunately more accurate measurements typically means more frequent measurements which increase the intrusiveness. We can ameliorate the effect of more frequent measurements by using shorter tests. While shorter tests will be less intrusive, TCP may never get out of slow start and the measured bandwidth will be lower than what is actually available.

We examine these questions by comparing the error and cost of measurements for different test times and test periods using a modified, experimental version of Gloperf. This version makes one test per minute between a fixed pair of hosts. A cycle of five test times are used: 1, 2, 5, 7, and 10 seconds. Hence, we collect five, interleaved time series of measurements, one for each test time with five minutes between tests. This allows us to examine the effect of test time. To evaluate the effect of test period, we will compare the error in this five-minute data with the error produced when the same data is subsampled to effectively produce a longer test period. For instance, we can subsample every twelfth point to effectively produce a one-hour test period. In the trace data that follows, we will use subsampling to produce 5, 10, 15, 30, 45, and 60 minute test periods.

Figure 1 shows the bandwidth measurements made with this experimental version of Gloperf between sleipnir.aero.org (near Los Angeles) and yukon.mcs.anl.gov (near Chicago) for approximately a 48 hour period over Los Nettos. While these time series are interleaved (and measure the network at slightly different times), the 1-sec. tests are clearly the lowest on average and longer test times produce a higher measured bandwidth.

The histograms of these time series in Figure 2 clearly show this trend, i.e., TCP coming out of slow start. These histograms also show that the 5, 7, and 10-sec. tests all share the same mode (even though the 10-sec. tests have the highest arithmetic mean by a slight margin). This indicates that tests longer than 5-sec. are not necessary on this link in accordance with the bandwidth-delay product governing TCP slow start.

To assess the "error" in these time series requires some notion of what the "true" values are compared to what Gloperf leads us to believe they are. This requires some form of *prediction model* based on Gloperf data. The simplest prediction model assumes that the last measurement predicts the next and the error is the difference between them. Hence, by using the *last measurement predictor*, we can assess the error by computing the Mean Absolute Error (MAE) and assess

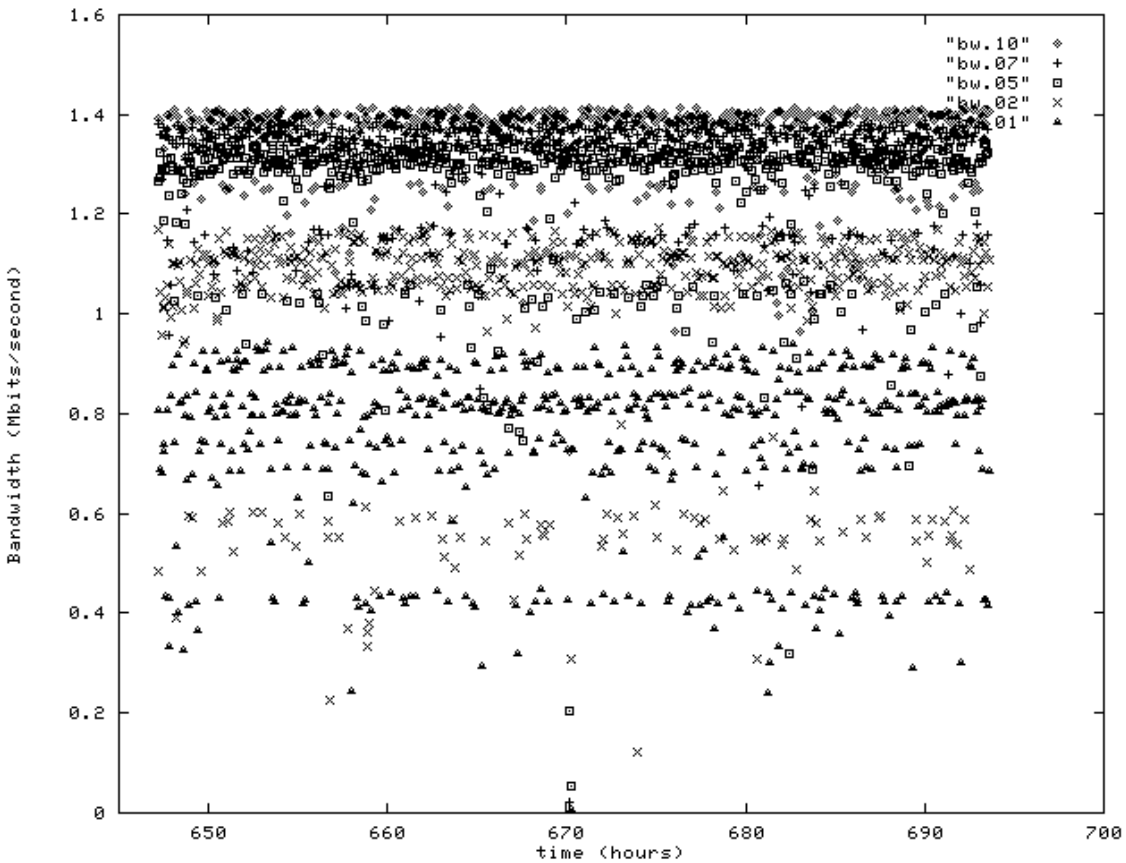

Figure 1: *Representative experimental bandwidth data with stationary means; sleipnir.aero.org to yukon.mcs.anl.gov.*

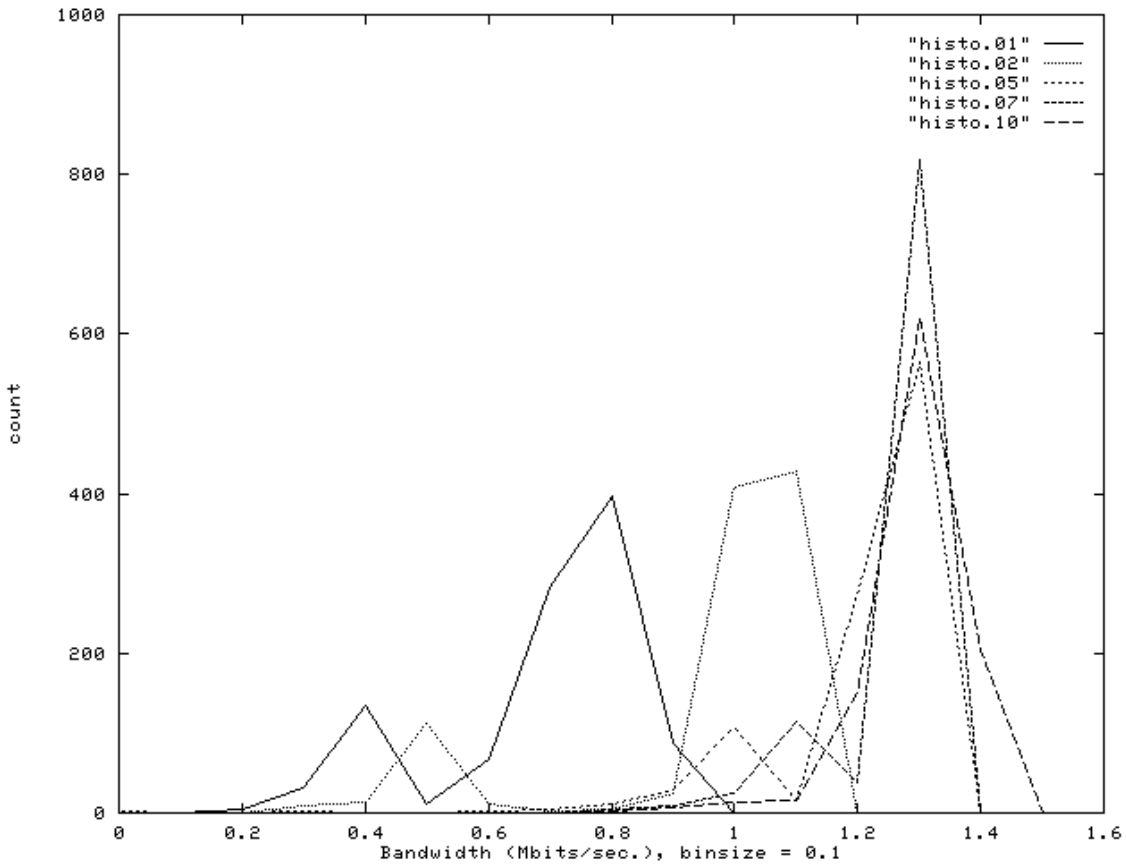

Figure 2: *Histograms of sleipnir-yukon data.*

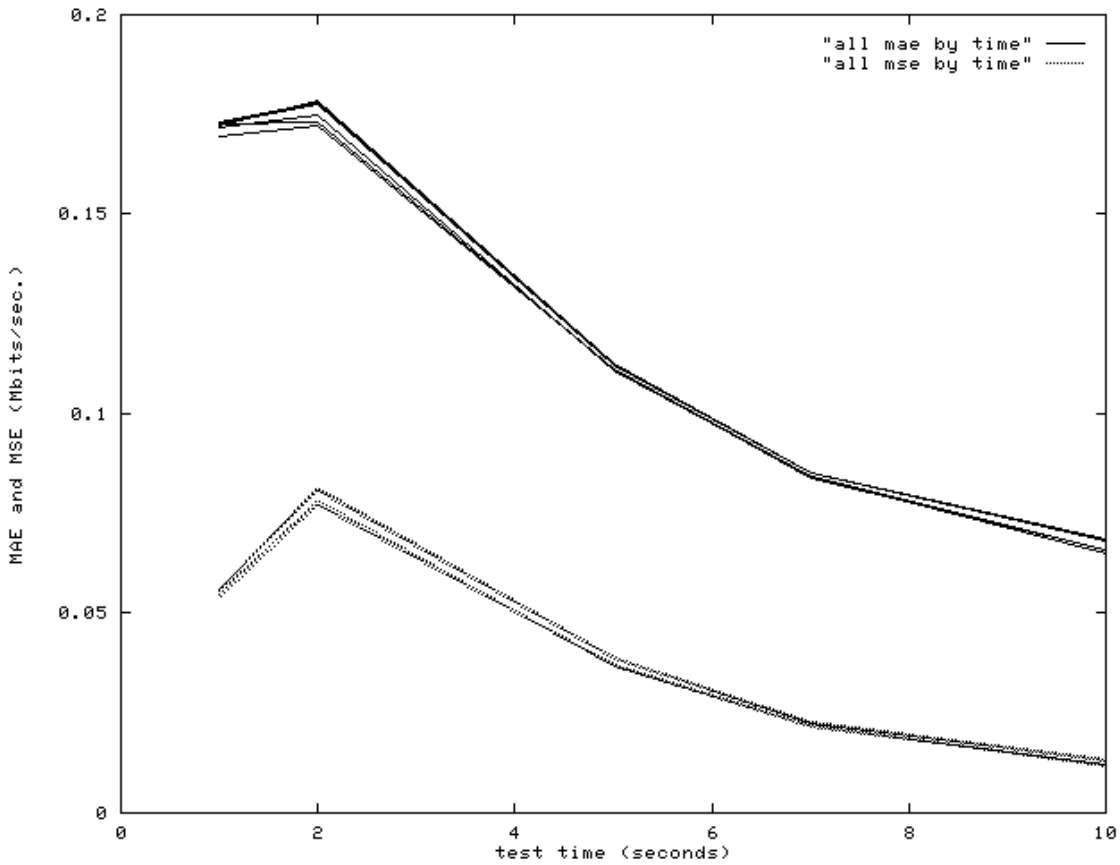

Figure 3: *sleipnir-yukon MAE/MSE as a function of test time and test period, plotted as a function of test time.*

the variability by comparing that with the Mean Squared Error (MSE). We can also compute the MAE and MSE for subsampled data. We note that when subsampling every n-th point, there are n different possible subsampled time series. Hence, for a given subsampling period, we compute the MAE and MSE for every possible series. While the distribution of these MAEs is of great importance, for the sake of simplifying the following graphs we only present their arithmetic mean. (MAE distribution is shown for the last, and longest, set of data.)

Using this method of computation, the MAE and MSE could be plotted as a surface with test time and test period as the independent variables. We will, however, present the data as a family of curves projected into one plane or the other. Hence, Figure 3 shows the MAE/MSE as a function of test time and Figure 4 shows the same data as a function of test period. Figure 4 shows that the test period has virtually no effect on the MAE. Testing once an hour is just as accurate as testing once every five minutes. This is, however, not surprising since the original data have relatively stationary means. In this situation, infrequent measurement is not only adequate but preferable.

The stability of these traces, however, allows us to see two interesting artifacts in Figure 3. First, the longer test times "average-out" some of the variability seen in the shorter test times resulting in a significantly lower MAE and MSE. Individual packet drops and retransmits also have less of an overall effect when the test time is longer. This is corroborated by the narrower histogram distribution when the test time is longer.

Second, we note an increase in the MAE for the two-second tests. Our current hypothesis concerning this is as follows. Packet loss is a discrete event and causes a quantized delta in the measured bandwidth. Hence, the measurements in Figure 1 actually occur in strata as shown by the multi-modal histogram distributions of Figure 2. At two seconds, TCP is in the process of leaving slow start which causes the strata to be spread out wider, producing a higher MAE. (This is

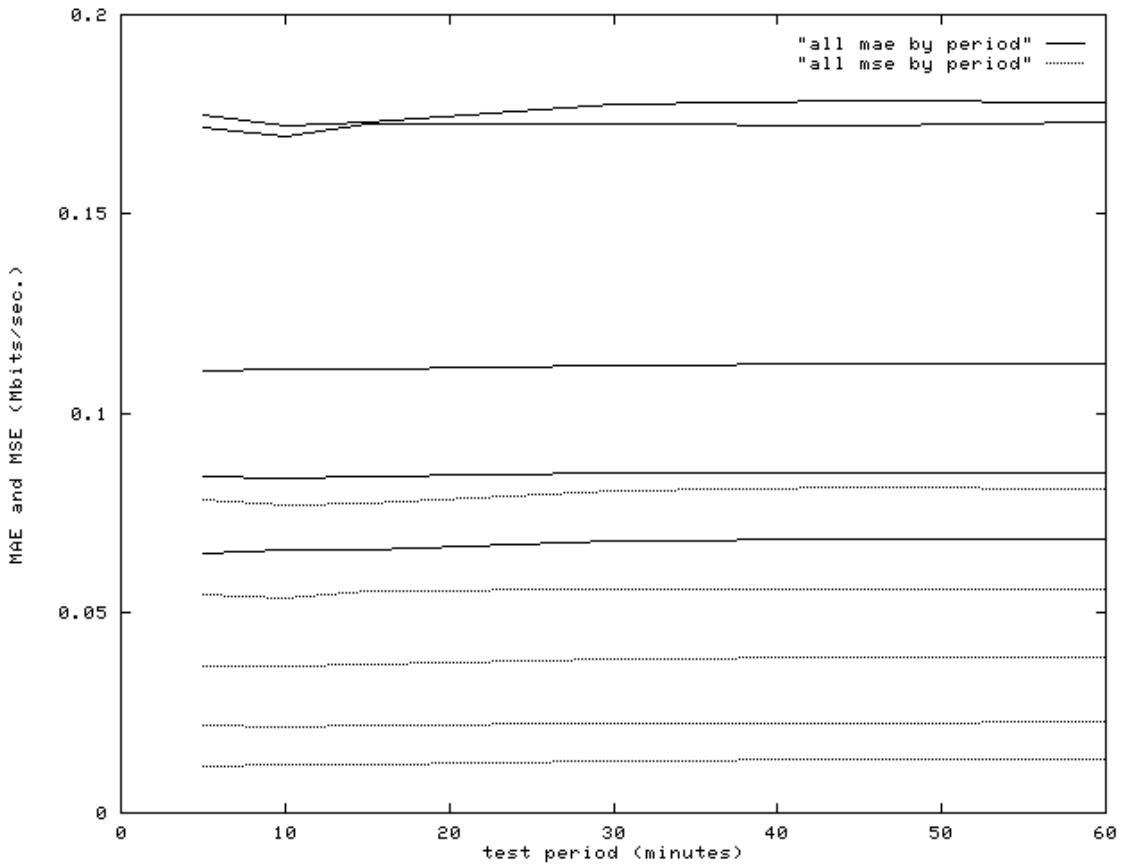

Figure 4: *sleipnir-yukon MAE/MSE as a function of test time and test period, plotted as a function of test period.*

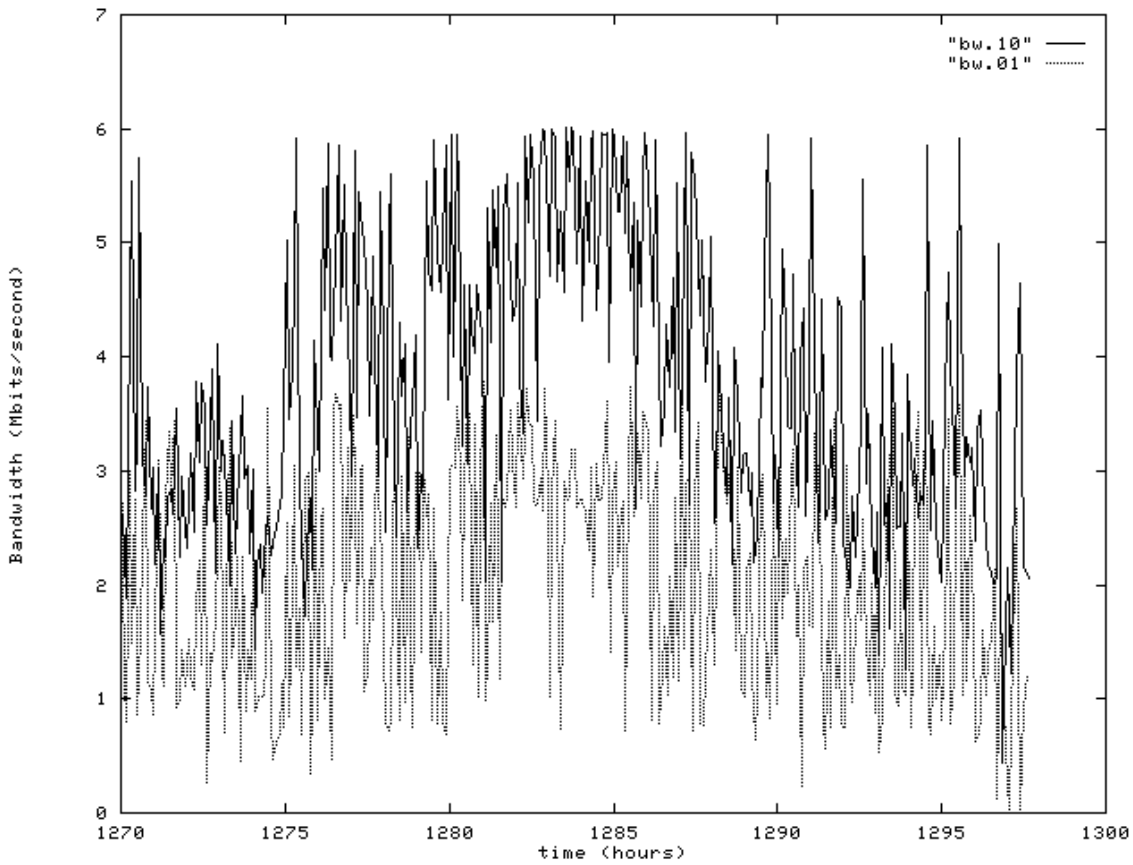

Figure 5: *Representative experimental bandwidth data with non-stationary means; jupiter.isi.edu to yukon.mcs.anl.gov.*

shown in the histogram data.) Once out of slow start with a longer test time, the strata are more narrowly constrained and an individual packet drop has less effect on the overall measured bandwidth, as one would expect.

Of course, not all links are so well-behaved. Hence, Figure 5 shows a similar experiment between jupiter.isi.edu (in Los Angeles) and yukon.mcs.anl.gov (near Chicago) over the vBNS that exhibits a much more volatile mean. For clarity, only the 1 and 10-sec. tests are shown. The histograms are shown in Figure 6.

Figure 7 shows the MAE and MSE for these traces by test time using the last measurement predictor. We note that not only are higher bandwidths achieved on the vBNS, the MAE and MSE are more than proportionally higher. As a sanity check, we employed a *random predictor*. This produces random bandwidth predictions with a uniform distribution across the range exhibited by the entire trace. Figure 7 shows that even in a "noisy" environment, infrequent measurements still produce a much better bandwidth estimate than random guessing.

One of the obvious drawbacks of the last measurement predictor is that an individual measurement may not be representative of the subsequent "true" bandwidth, i.e., the measurement happened to hit a transient peak or trough in the trace. Indeed, it is possible to consider infrequent measurements not as part of a time series but rather as members of a population sample. This means that a simple arithmetic mean might predict bandwidth just as well as the last measurement, if not better. In a long-lived operational environment, however, an extremely large number of samples would eventually be collected and a new measurement would never perceptibly change the mean. Hence, much like the venerable Unix load average, a running exponential average could be used to track the current "load" based on a weighted sum of the current measurement and the previous "load". This allows current peaks and troughs in the trace to be weighted against recent past measurements with an exponential decay.

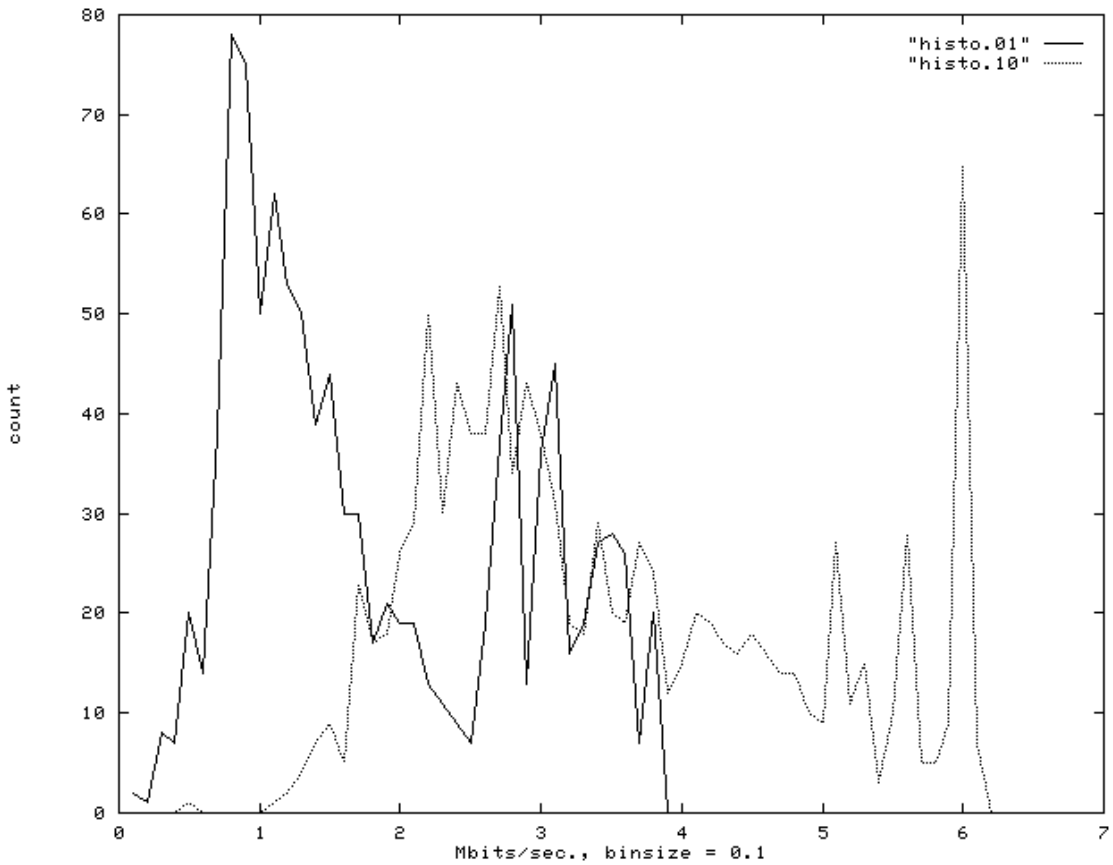

Figure 6: *Histogram of jupiter-yukon data.*

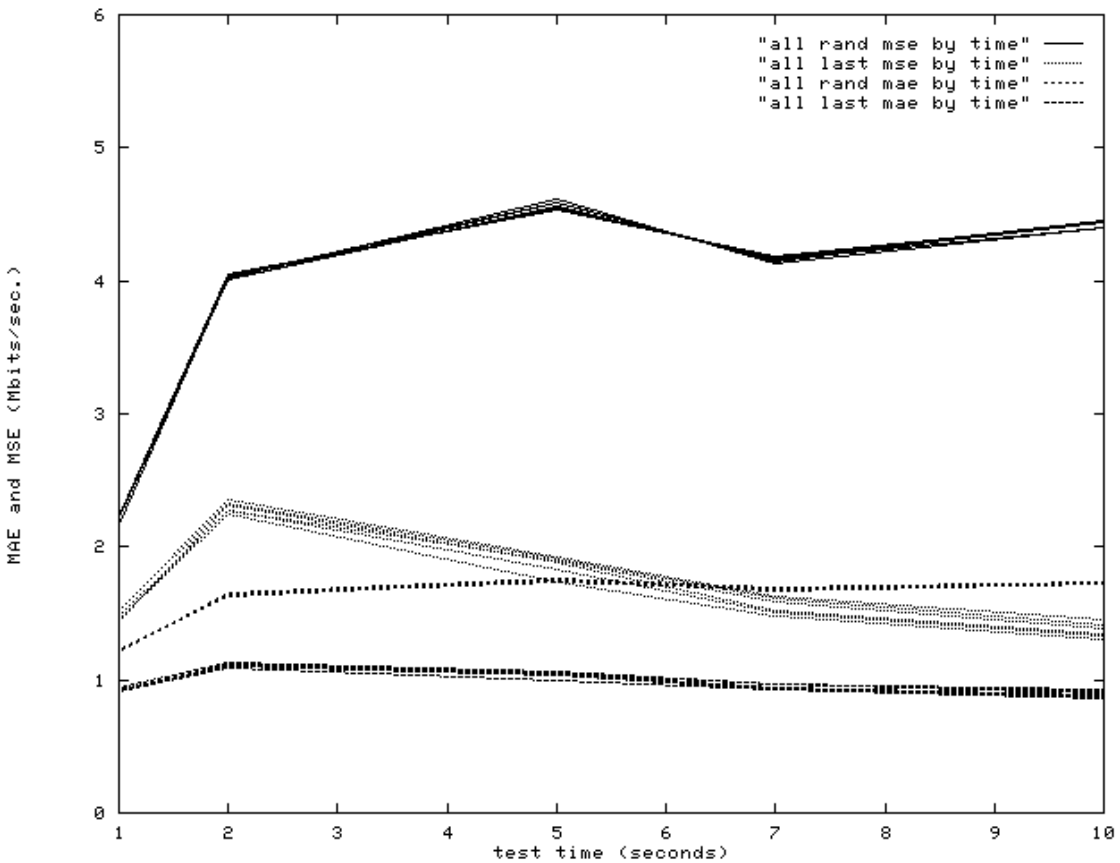

Figure 7: *jupiter-yukon MAE/MSE by test time, using a last-measurement predictor and a random predictor.*

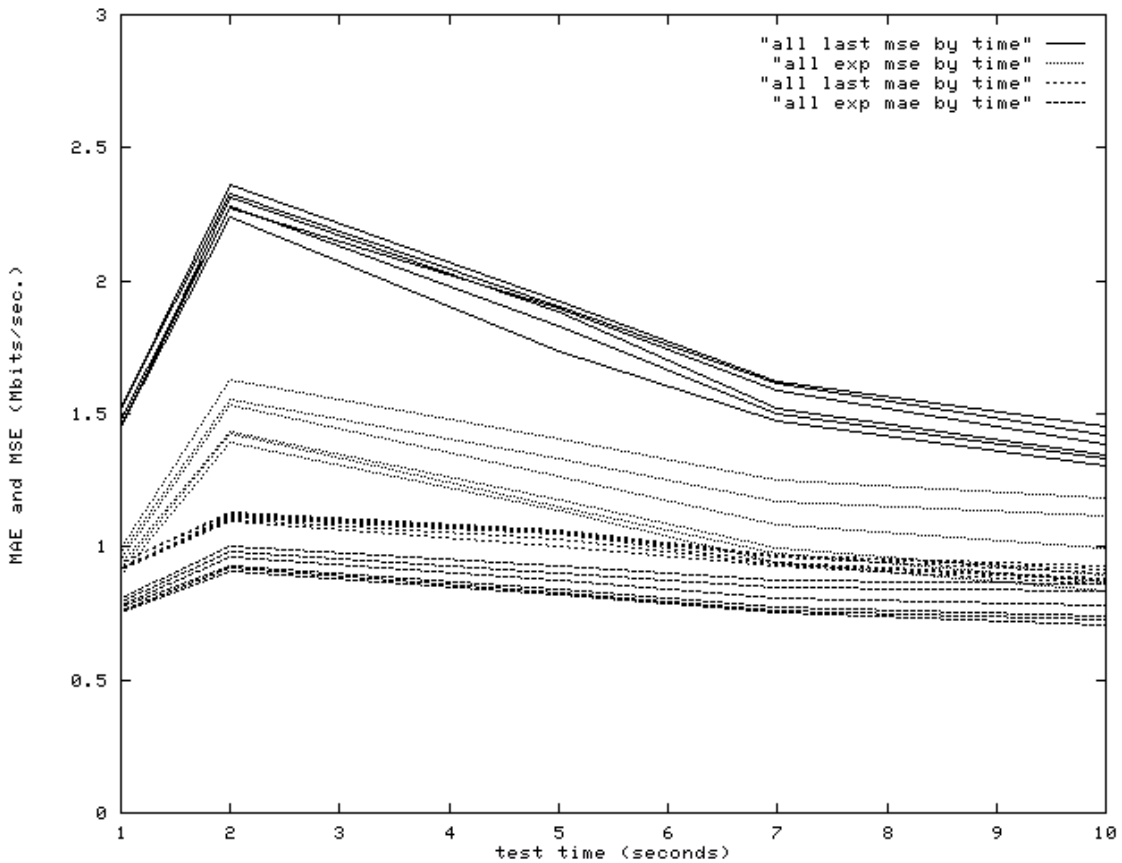

Figure 8: *jupiter-yukon MAE/MSE by test time, using a last-measurement predictor and an exponential average predictor.*

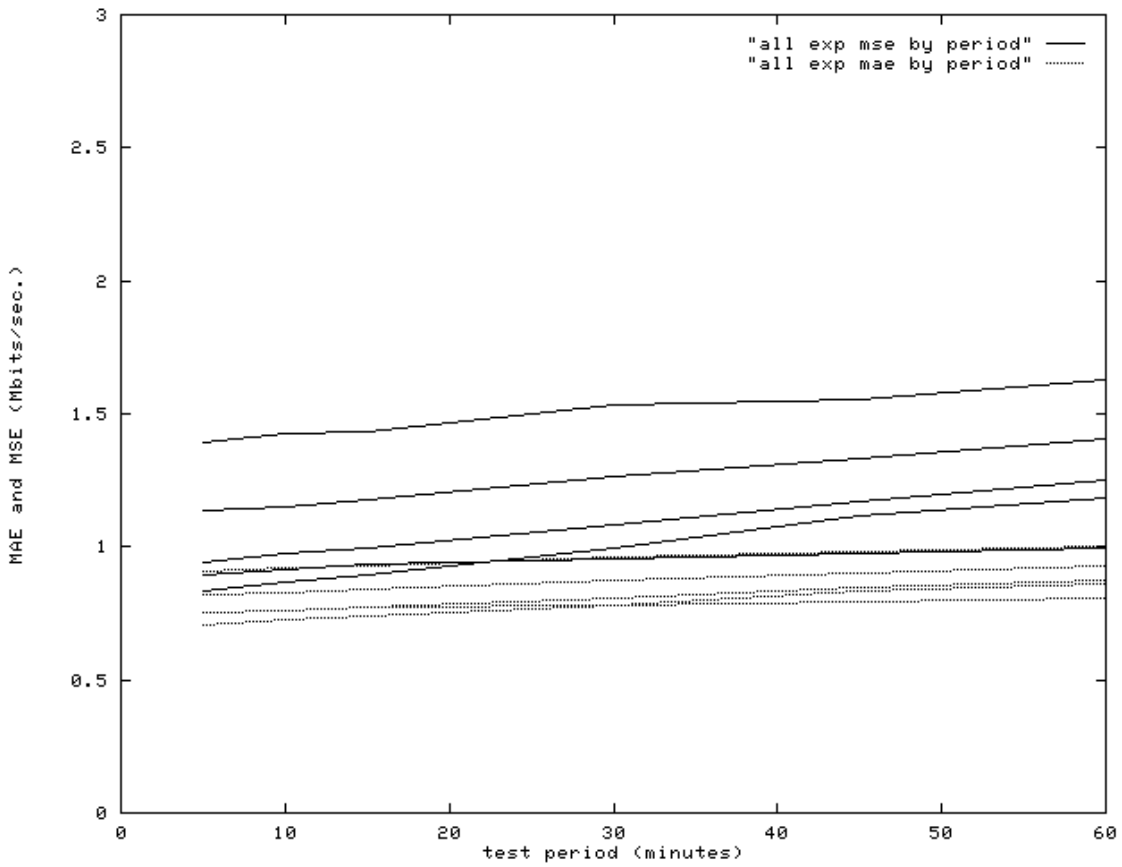

Figure 9: *jupiter-yukon MAE/MSE by test period, using an exponential average predictor.*

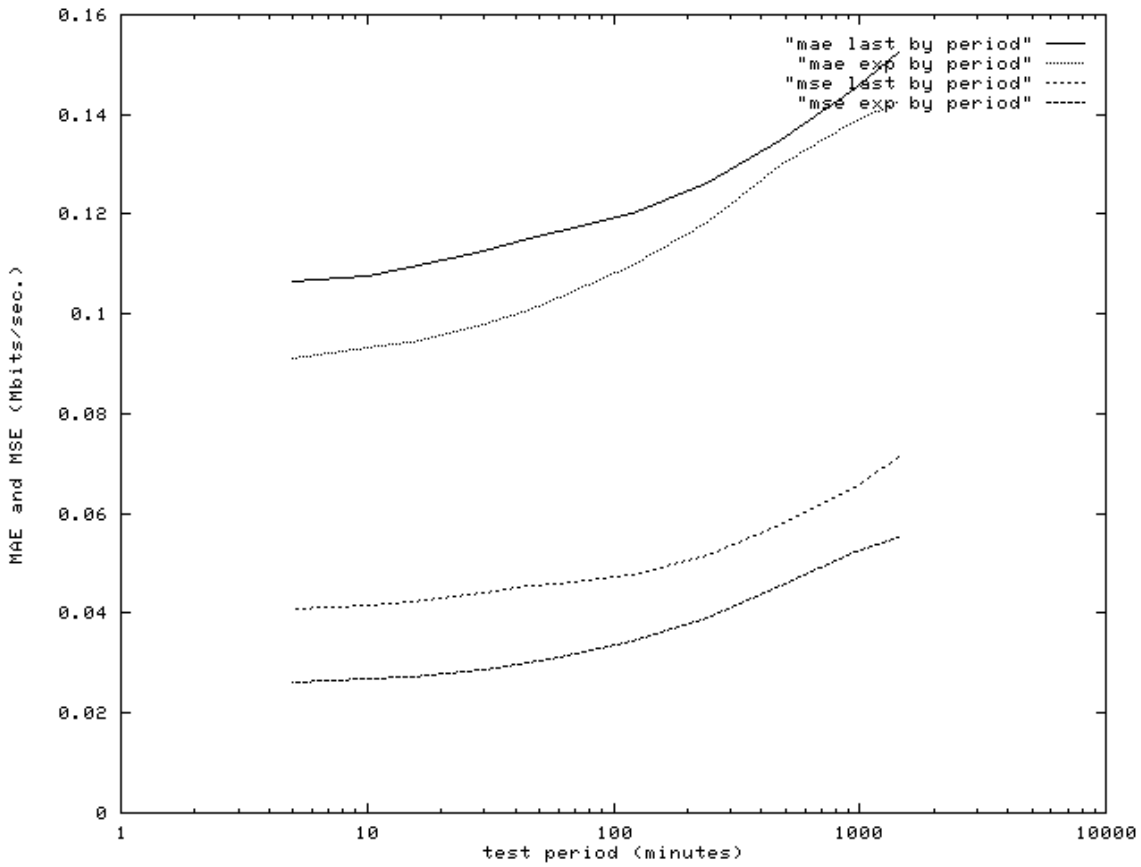

Figure 10: *MAE/MSE of subsampled NWS trace by test period, using a last-measurement predictor and an exponential average predictor.*

Figure 8 shows the MAE and MSE for both the last measurement predictor and an exponential average predictor with a weight of 0.35 on the most recent measurement. Work with the Network Weather Service [16] has shown that simple forecasting techniques such as exponential smoothing can improve prediction accuracy dramatically over the simple use of monitor data alone. Our work with Gloperf confirms this result emphasizing the need for forecasting in grid computing settings. In this case, exponential smoothing shows a clear improvement in the prediction error, even for one-hour test periods. Figure 9 shows the same exponential average data plotted as a function of test period. Once out of slow start, the MAE only rises approximately 10% from testing every five minutes to testing once an hour, even in a typical environment.

What happens, however, when the test period rises to as much as once every 24 hours? To examine this question, we used a trace generated by the Network Weather Service that contained 69751 high-frequency tests over a 28.8 day period. We subsampled this trace to produce a trace with 7163 tests at a test period of 300 seconds over the same 28.8 days. Figure 10 presents the MAE and MSE of this trace for test periods of 5, 10, 15, 30, 45, 60 minutes and 2, 4, 8, 12, 24 hours. Beyond one hour, the MAE does start to rise more quickly. Nonetheless, the average worst error measuring once a day only amounts to less than 20% of the actual bandwidth observed.

One interesting aspect of the trace data is that interleaved time series do not produce the same MAE and MSE error readings. Indeed, two series shifted only by one measurement can generate different overall forecasting errors. We do not, as yet, understand fully the nature of this difference, although we conjecture that it may be due to "hidden" periodicities in the series. A frequency analysis does not expose any such periodicities, however, leading us to consider other possible causes. Therefore, we show the average MAE and MSE for each point in Figure 10.

To understand the extent of this variation, we histogram the MAE and MSE readings for adjacent subsamples in

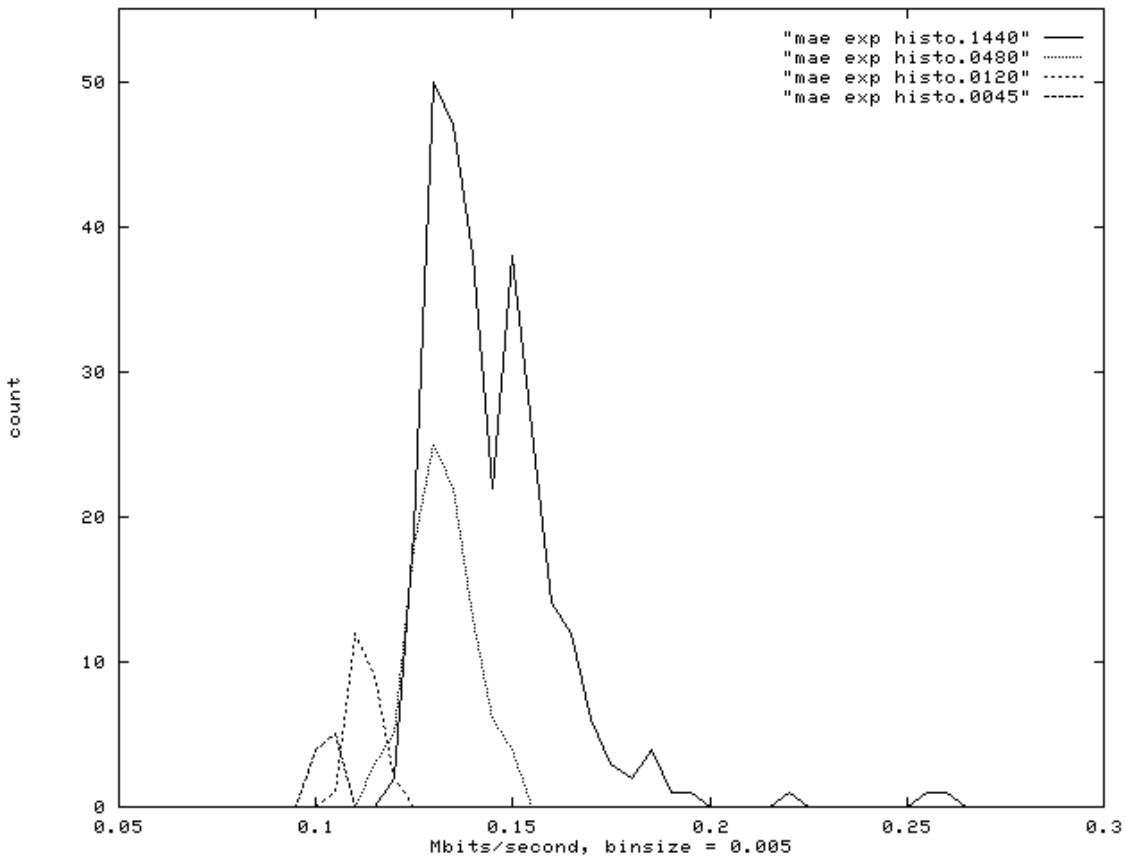

Figure 11: *Histogram of exponential predictor MAEs from subsampled NWS trace.*

Figure 11. Despite the variability, the worst-case sequence with a one-day measurement period produced an error of only slightly worse than 25% of the actual bandwidth observed. Also note that the 24 hour period exhibits the widest degree of variability leading is to conjecture that cyclic frequency components are the cause of the variability.

Performance Discussion.

While these data are only representative, they indicate that longer test periods to not erode accuracy as measured by the MAE to a point where the data is completely useless. (We are collecting longer Gloperf traces from a variety of national and international hostpairs to further corroborate our results.) While longer test times tend to average-out variations, they also increase the intrusiveness as measured by the total number of bytes transferred per hour. We note that the overhead of Gloperf in total number of bytes transferred per hour is linearly related to the test period. Hence, going from a 5 minute to 60 minute test period represents a 12x reduction in testing data volume. Going from 1 hour to 24 hours represents a 24x reduction. We note that on fast, under-utilized links, Gloperf can push a larger data volume than is necessary. As an enhancement, Gloperf could possibly limit test time based on the bandwidth-delay product for the link under test. This would limit the data volume transferred in a test but also ensure that TCP has come out of slow start.

This brings up a larger issue: what is the best metric for "intrusiveness"? The number of bytes transferred? The time spent sending additional traffic? The bandwidth actually denied to other flows? The congestion precipitated at routers along the path? The answer is "all of the above". Unfortunately not all of these metrics are easy to capture. A low-level infrastructure that could capture items such as bandwidth denied or congestion at every router on a path would be difficult to implement and deploy. While such an infrastructure could produce useful insights, most cost comparisons will continue to be done with metrics that are approximate and easy to obtain.

We also reiterate that Gloperf is primarily a sensor and collection mechanism; it does not "contain" any prediction models itself. We have, however, used simple last measurement and exponential average predictors to establish that infrequent measurements, as provided by Gloperf, can be used to "predict" network performance with an acceptable amount of error. We note that other, more accurate prediction models are certainly possible, e.g., prediction of diurnal and even weekly cycles, that are based on historical data and other known facts about a particular link.

5 Conclusions and Future Work

We have presented a very simple network performance monitoring system for grid computations. Infrequent measurements are very unintrusive and the accuracy does not degrade to where they cannot be used for purposes such as initial resource discovery and allocation in a grid computing environment. Infrequent measurements will, of course, completely miss high-frequency volatility in network performance. Applications that cannot tolerate such volatility can attempt to adapt by using self-measurement, or high-frequency measurements as in the NWS, or ultimately relying on a bandwidth reservation mechanism that controls the volatility directly.

While the initial deployment of Gloperf has been kept simple, some further developments are possible. On-demand measurements would allow applications to probe network resources to get very fresh data. Automatic group discovery is another topic. Contention has not been a problem with such infrequent measurements. If it ever becomes a problem, however, some form of arbitration will have to be implemented. Finally we recognize some data usability issues. Organizing Gloperf data based solely on source host is not appropriate for all situations. To alleviate this, we plan to implement different LDAP-based *service providers* that are logically indexed as being part of the MDS but are free to acquire and manage the raw data and any derived data in the most appropriate way for the service they provide.

References

[1] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G Shao. Application level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing 1996*, 1996.

[2] R. Carter and M. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. Technical report, Boston U., 1996. Technical Report TR-96-007. Available from `http://cs-www.bu.edu/students/grads/carter/papers.html`.

[3] R. Carter and M. Crovella. Measuring bottleneck link speed in packet-switched networks. Technical report, Boston U., 1996. Technical Report TR-96-006. Available from `http://cs-www.bu.edu/students/grads/carter/papers.html`.

[4] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proceedings 6th IEEE Symp. on High Performance Distributed Computing*, pages 365–375, 1997.

[5] Cooperative Association for Internet Data Analysis. Caida. Technical report, CAIDA, 1999. Available from `http://www.caida.org`.

[6] T. Howes and M. Smith. *LDAP: Programming Directory-enabled Applications with Lightweight Directory Access Protocol*. Macmillan Technical Publishing, 1997.

[7] V. Jacobson. A tool to infer characteristics of internet paths. Technical report, Lawrence Berkeley Lab, 1997. Available from `http://ftp.ee.lbl.gov/pathchar`.

[8] R. Jones. Netperf. Technical report, 1999. Available from `http://www.netperf.org/netperf/NetperfPage.html`.

[9] B. Lowecamp et al. A resource query interface for network-aware applications. *7th IEEE Symp. on High Performance Distributed Computing*, pages 189–196, August 1998.

[10] M. Mathis and J. Mahdavi. Diagnosing internet congestion with a transport layer performance tool. *Proc. INET '96*, 1996.

[11] K. Obraczka and G. Gheorghiu. The performance of a service for network-aware applications. *2nd Sigmetrics conference on parallel and distributed tools*, August 1998.

[12] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An architecture for large-scale internet measurement. *IEEE Communications*, 36(8):48–54, August 1998.

[13] P. Stelling, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski. A fault detection service for wide area distributed computations. In *7th IEEE Symp on High Performance Distributed Computing*, pages 268–278, 1998.

[14] A. Su, F. Berman, R. Wolski, and M.M. Strout. Using apples to schedule a distributed visualization on the computational grid. November 5 1998. Available from `http://www.cs.ucsd.edu/groups/hpcl/apples/hetpubs.html`.

[15] The Globus Team. The Globus Metacomputing Project. 1998. Available from `http://www.globus.org`.

[16] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1998. also available from http://www.cs.utk.edu/ rich/publications/nws-tr.ps.gz.

[17] R. Wolski, N. Spring, and H. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computing Systems*, 1998. Available from `http://www.cs.ucsd.edu/users/rich/papers/nws-arch.ps`.