Direct Queries for Discovering Network Resource Properties in a Distributed Environment

Bruce Lowekamp [a,*] David O'Hallaron [a,b] Thomas Gross [a]

[a] *Computer Science Department*
Carnegie Mellon University
Pittsburgh, PA 15213
[b] *Electrical and Computer Engineering Department*
Carnegie Mellon University
Pittsburgh, PA 15213

The development and performance of network-aware applications depends on the availability of accurate predictions of network resource properties. Obtaining this information directly from the network is a scalable solution that provides the accurate performance predictions and topology information needed for planning and adapting application behavior across a variety of networks. The performance predictions obtained directly from the network are as accurate as application-level benchmarks, but the network-based technique provides the added advantages of scalability and topology discovery.

We describe how to determine network properties directly from the network using SNMP. We provide an overview of SNMP and describe the features it provides that make it possible to extract both available bandwidth and network topology information from network devices. The available bandwidth predictions based on network queries using SNMP are compared with traditional predictions based on application history to demonstrate that they are equally useful. To demonstrate the feasibility of topology discovery, we present results for a large Ethernet LAN.

Keywords: distributed systems, computational grid, network performance analysis, network topology

AMS Subject classification: Primary 68M14, 68M10; Secondary 68M20

1. Introduction

Network-aware programming is emerging as an effective way of adapting to fluctuating network conditions. Providing the information of which applications should be "aware," however, has proven to be a significant challenge to developing such applications [2]. User-level benchmarks that record the performance of data transfers across the network provide some information, but suffer from poor scaling and the inability to determine network topology.

Scaling is obviously important in distributed system design. A network performance prediction system must have the ability to give predictions for any combination of machines selected from a large set of possible machines at many sites across the network.

Topology information is extremely important for performance-based machine selection. For example, without topology information, it is impossible to determine when multiple communication paths within the same application will be sharing a single network link. Failure to consider this intra-application sharing can result in an overestimation of the network performance an application can achieve.

An alternative to benchmarking is obtaining performance information through direct queries to the components making up the network. We show that this direct query approach is not only as accurate as benchmarking but also can lower the load imposed on the network and provide previously unavailable information about network topology. The Simple Network Management Protocol (SNMP)[5] provides an interface to current networking hardware through which queries can be made to obtain information about the hardware's status.

The remainder of this paper will discuss our results with using SNMP to obtain network information. Section 2 describes several techniques for performance prediction, including benchmark- and network-based ap-

* Corresponding author, lowekamp@cs.cmu.edu

proaches. Section 3 provides an overview of the structure of SNMP and how it can be used for resource status discovery. Sections 4, 5, and 6 analyze the accuracy of SNMP for obtaining predictions of available bandwidth. Section 7 discusses how to extract topology information from the network components. Section 8 discusses changes needed in the network infrastructure, including SNMP, to make it easy for distributed application programmers to use direct network queries to take advantage of the information already stored in the network.

2. Bandwidth prediction techniques

Network performance predictions are needed by applications for uses such as machine selection and setting application quality parameters. This section discusses three different techniques for performance prediction and introduces a model that describes these techniques. The three techniques represent a continuum of options, from the most straightforward prediction based on the application itself to the least straightforward benchmark-based prediction. This section ignores all performance factors except for network bandwidth. In cases where network bandwidth is not the predominant bottleneck, other resource information must be considered.

All three bandwidth prediction techniques rely on the same basic time series prediction models, which use a series of measurements to make predictions of future behavior. The difference between the three techniques is what measurements are taken and how they are converted to a prediction of application performance. Ideally, the series of measurements is taken by an independent daemon that collects the data for later use when a user wishes to run an application. A mathematical model is fit to the series. When a user requests a prediction, future performance is extrapolated from the model that has been fit to the past data. Selection and use of time series models has been dealt with by many authors [3,8,20]. In our notation, time series models are indicated by a t subscript on the measurement that is used for the series.

2.1. Application-based

The most straightforward measure of an application's performance is obtained by actually running the application on the network. Similarly, the most straightforward prediction of an application's future performance on that network is obtained using the application's performance history on that network to predict its future performance. The performance of an application A running on a network $\mathcal{N}$ is denoted $A(\mathcal{N})$. The time series model $A_t(\mathcal{N})$ can be used to predict the application's performance on the network.

Unfortunately, the many combinations of applications, parameters, and resource selections make gathering enough application history information to provide useful predictions infeasible. For this reason, other prediction techniques must be considered.

2.2. Benchmark-based

Benchmarking solves many of these problems by using a small set of representative applications, called benchmarks or probes, to predict the performance of many applications. The performance of the benchmarking application is denoted $B(\mathcal{N})$. Again, a time series of benchmarks can be used to form a prediction, $B_t(\mathcal{N})$, of how the benchmark B will perform on the network $\mathcal{N}$ in the future.

The challenge with using benchmarks for performance predictions is the mapping from benchmark performance to application performance. One method is to make the assumption that the relative performance of the application and benchmarks are the same, so the best connection for the application is assumed to be the same as the best connection for the benchmark. This approach is often useful for parallel applications where the only concern is moving the data as quickly as possible.

The lack of quantitative information about the application's performance, however, prevents this technique from being useful in many situations. It does not answer the question of which connections are sufficient for the application's needs, nor does it provide information necessary for setting application quality parameters. Quantitative information is needed for these decisions.

Benchmarking can be used to provide quantitative information, and for some applications, a benchmark will perform similar operations so that the results can be used with a simple rescaling. In other cases, such as using a TCP-based benchmark to predict the performance of a multimedia application that can handle loss, it is necessary to develop a model of the network con-

ditions that caused the benchmark's performance and then to determine how the application will perform under those same conditions. A mapping function converting the predicted performance of the benchmark to a model of the network can be written as $M(B_t(\mathcal{N}))$. A prediction of application performance based on this network model can be written $A_n(M(B_t(\mathcal{N})))$, where the subscript n is used to denote a performance prediction for the application A based on a network resource model.

2.3. Limitations of these techniques

Application- and benchmark-based predictions rely on sending data across the network to obtain the measurements needed for performance prediction. These methods have the advantage of treating the network like a black box, avoiding the complexities inside the network. But there are a number of limitations that are shared by these techniques.

2.3.1. Scaling

Sending data between two machines is an excellent way of measuring the network's performance between those two machines. Unfortunately, applications that can choose between several machines require more information. Examples include:

- selecting one of several machines for a long-running application,
- selecting the best n machines for a parallel computation,
- a real-time scheduling application that forwards tasks to the best-available machine, and
- selecting machines in a collaborative environment where information is needed for choosing servers to work with several desktops distributed across the network.

Each of these situations requires knowledge of the network performance between more than a single pair of machines. The minimal amount of information required to solve these problems is the network performance between each pair of potential servers and between each desktop and all of the servers. Because modern environments allow desktop machines to function as servers, these problems can only be solved with knowledge of the communication performance between all pairs of machines. If P is the number of machines in the system, this is an $O(P^2)$ problem. Even for the number of machines that might be found in a small department, perhaps $P = 50$, taking a measurement of network performance every five minutes would require almost ten measurements per second.

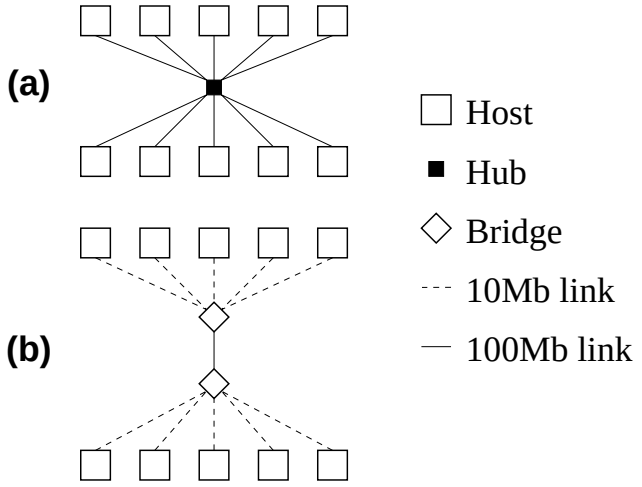

Figure 1. Two networks for which performance predictions cannot be made without knowledge of the topology.

2.3.2. Invasiveness

Another fundamental problem of measuring network performance by sending data across the network is that it quite naturally disturbs the system it is measuring. Obtaining a prediction that 10Mbps is available on the network is useless if the measurement system is using that 10Mbps 25% of the time.

2.3.3. Topology

Finally, none of these measurement techniques provide information about topology. At first glance, knowing a network's topology may seem useful only as a way of improving the scalability of the predictions. However, it is much more important for accurate predictions of the performance of parallel applications.

Consider the simple network shown in Figure 1(a). Assuming no other traffic, each pair of machines in this network will see the full 100Mbps bandwidth available in the network. However, the hub connecting the machines makes the entire network a single data link; if all of these machines are used to run a single application, the available bandwidth per machine will be 10Mbps. On the other hand, if the machines were connected via a bridge, then the full 100Mbps bandwidth would be

available to each machine.

Manually providing the topology to the benchmarking tool allows for application-level sharing to be predicted and could solve the problem with this particular example. However, even with knowledge of the topology, it is sometimes impossible to predict the performance of an application with simple end-to-end benchmarks. In the network depicted in Figure 1(b), there is no way to determine whether the bandwidth of the central link is 10 or 100Mbps, because it is irrelevant to a single network connection. However, once again, a parallel application can be significantly impacted by this difference.

Both of these problems can be overcome by performing multiple simultaneous benchmarks to determine whether there are correlations in the performance of separate connections. Unfortunately, this also raises the complexity of the technique to an infeasible $O(P!)$.

2.4. Network-based

Rather than obtaining the performance measurement from an application, another approach is to obtain it by querying the network itself. This allows an exact picture of the status of the entire network to be obtained. Network topology can be obtained this way, and the cost is linear in the size of the network. A snapshot of the network $\mathcal{N}$, consisting of the status of all parts of the network at the same instant, is denoted N. Using a history-based prediction of the network snapshot, N_t, an application's performance can be predicted as $A_n(N_t)$.

The three techniques described here for obtaining a prediction of an application's future performance running on $\mathcal{N}$, denoted $\hat{A}(\mathcal{N})$, are described by the following equations:

$$\hat{A}(\mathcal{N}) \approx \begin{cases} A_t(\mathcal{N}) & \text{Application-based} \\ A_n(N_t) & \text{Network-based} \\ A_n(M(B_t(\mathcal{N}))) & \text{Benchmark-based} \end{cases}$$

$\mathcal{N}$	real network
N	network status snapshot
A	application
B	benchmark
$A(\mathcal{N})$	performance of A running on $\mathcal{N}$
$A_t()$	time series performance prediction of A
$A_n()$	network model performance prediction of A

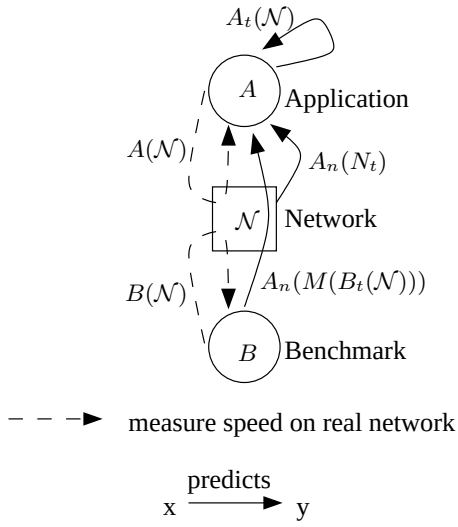

Figure 2. Conceptual diagram of options for prediction. The dashed arrows illustrate the actual application being run on the network as a probe of its status. The solid arrows illustrate the conceptual paths taken to predict an application's performance using data obtained by running the application itself, by obtaining information directly from the network, and by using a benchmarking program to determine the network's performance.

$M()$	mapping function inferring network status from benchmark performance

Figure 2 conceptually illustrates how these techniques interact with the network to predict application performance.

The network-based technique offers several improvements over the use of benchmarking or application history.

- Direct measurement of the network's status allows the performance of different network operations to be predicted, without the need for many different types of benchmarks.
- The network's topology is acquired directly from the network, allowing application-level sharing to be predicted.
- Direct network queries require only $O(|\mathcal{N}|)$ operations, imposing significantly lower load on the network than $O(P^2)$ benchmarks to obtain the same information.

The major challenge of using network-based predictions of application performance is the necessity of developing the performance prediction function $A_n()$,

which maps network status to the application's performance. Our current approach to this problem is to establish a prediction function by querying the network, then running and measuring the application. Once enough measurements are taken, a predictor can be built that takes the bottleneck link of the network as input and predicts the application's performance over that link.

3. SNMP overview

SNMP was designed to allow network managers to remotely observe and adjust network components. It defines the structure of and operations on a database that is stored on each network component. The database is organized hierarchically, with portions reserved for various standards bodies and vendors. Components are free to implement only those portions of the hierarchy that are desired. Each portion of the hierarchy is specified by a document referred to as a Management Information Base (MIB). Although it is more correct to refer to only the database protocol as SNMP, in common usage SNMP is used to describe the collection of MIBs as well as the protocol. We follow common usage unless distinctions are needed for clarity. For more information about SNMP and MIBs, many books have been written for use by network managers [18].

3.1. MIB-II

RFC1213 describes the standard MIB, called MIB-II [10]. It is intended to describe essential information needed for all network components—including hosts, routers, and bridges. It provides information about components' offered services and networking hardware. It also provides statistics and information about major networking protocols, including IP, TCP, UDP, and SNMP.

Two parts of this MIB are of interest to us. The first is the interface table. The row of data for each interface provides its maximum data rate as well as octet counters, which indicate the number of bytes the interface has sent and received. This information allows the available bandwidth on the link attached to that interface to be determined.

Another useful component of this MIB is the IP routing table, which indicates the route a device will use to send IP packets to their destination. This is the most important information for determining a network's topology.

3.2. BRIDGE-MIB

The second most important MIB is the BRIDGE-MIB [6]. This MIB provides information about the status of an Ethernet bridge, which is used to forward packets between different portions of a LAN. The interesting part of this MIB is the forwarding database, which stores the port used to reach each of the Ethernet addresses the bridge has seen. Because bridges operate transparently, making queries from this MIB on each bridge is the only way to obtain the information needed to construct the topology of an Ethernet LAN.

4. Testbed verification

While there are clear advantages to the network-based technique, making predictions about end-to-end operations using low-level information is inherently difficult [16]. We have verified the network-based technique against an application-based technique. If the two techniques offer similar accuracy, the scaling and efficiency advantages of the network-based method make it the better choice for performance prediction.

These experiments were performed on a dedicated testbed where the conditions could be controlled to represent a wide variety of congestion levels. Because of the breadth of conditions experienced on networks, it is important to test prediction techniques at all levels of congestion [13].

4.1. Experimental setup

The network configuration used in the experiments is shown in Figure 3. The "application" used was a simple 1MB data transfer from A to B using TCP. Every 15 seconds, SNMP was used to measure the available bandwidth on all segments of the path between A and B, followed immediately by the application's data transfer. To measure available bandwidth over different averaging periods, the SNMP packet counts were obtained 5, 3, 0.5, and 0 seconds prior to the TCP message.

The 1MB data transfer would be a typical benchmark. However, for this experiment, it is considered an application because it is being used to predict its own, rather than other applications', performance. A

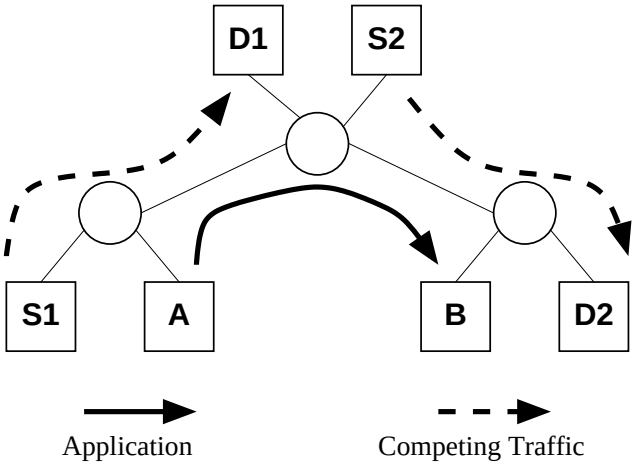

Figure 3. Topology of the testbed used for the prediction experiments. All links are 100Mb. The hosts are 300Mhz DEC Alphas and the routers are Cisco 7206 routers.

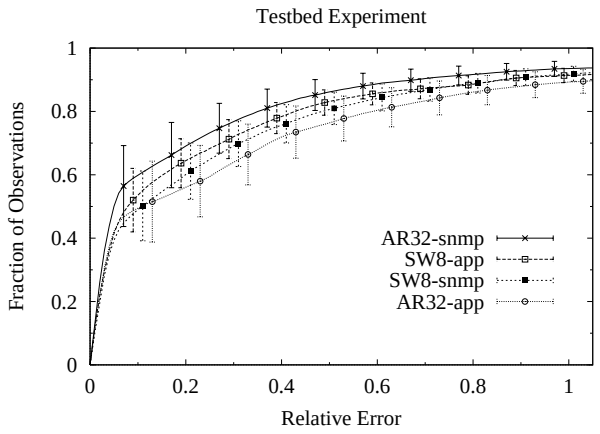

Figure 4. Cumulative relative error distributions for application- and network-based prediction in the testbed experiment. Both 32nd-order autoregressive (AR32) and 8 observation sliding window mean (SW8) predictions are shown. The SNMP available rate was averaged over 3 seconds for each observation. 95% confidence intervals are shown.

real application would also involve computation, which is being ignored for the purpose of this paper.

Synthetic traffic was inserted onto the network between S1 and D1, and S2 and D2, resulting in two congested links competing for bandwidth with the application. The competing traffic was generated using fractional Gaussian noise, a method described by Paxson for representing realistic aggregate traffic encountered on networks [12]. The average rate of competing traffic on each link was chosen between 0Mbps and 100Mbps (link capacity) and changed an average of every 10 minutes.

4.2. Experimental results

Over 65,000 observations were taken during the experiments. To determine the accuracy of the two prediction techniques, 30 sets of 1500 consecutive observations were chosen at random from the experiment. The first 1000 were used to fit the time series model. The prediction technique was then tested over the next 500 observations. The model was refit for each additional observation, so the time series model was only used to predict one observation interval ahead. Each prediction was compared with the next actual observation.

The implementation of the time series predictors that were used is described by Dinda and O'Hallaron [8]. The autoregression (AR) and sliding window average (SW) prediction models were used. Wolski examined several prediction models and found these two to be useful for network performance prediction [20].

The application prediction was performed on the series of times recorded for the 1MB data transfer. The relative error between each step-ahead prediction, $A_t(\mathcal{N})$, and the next actual observation, $A(\mathcal{N})$, was recorded.

For the SNMP-based predictions, the time series models were applied to the series of SNMP available bandwidth measurements. $A_n()$ was a piecewise linear interpolation initially built for each data set with the 1000 training observations. As the experiment proceeded, each subsequent observation was added. In a real system, the calibration would be done less often. The relative error was calculated between $A_n(N_t)$ and $A(\mathcal{N})$.

A comparison between the relative errors is shown in Figure 4. The important observation is that there is little difference between the application- and network-based techniques. This leads us to conclude that the network-based prediction technique can be used to provide network predictions with accuracy comparable to application-based techniques.

5. Simulated verification

Because it is not possible to reproduce all ranges of network behavior on a testbed, simulation was used to explore a wider range of parameters. The simulator allows the replay of actual network traces and simple selection of network bandwidth, producing realistic traffic

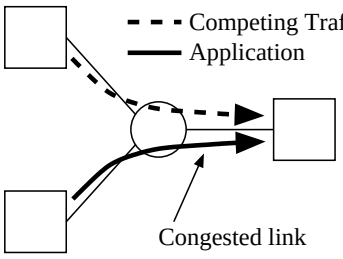

Figure 5. Simulated network topology. The marked link is the only bottleneck, resulting in congestion at the router that connects the three hosts.

in a flexible environment.

The NS simulator is one of the most widely used network simulators available today [1]. This experiment used a limited subset of the NS simulator's capabilities. The background traffic was generated as UDP packets replaying a tracefile collected from a real network. The application's traffic was a simple Reno TCP connection.

The traces used were obtained from the Passive Measurement and Analysis project of the Network Analysis Infrastructure being developed by the National Laboratory for Applied Network Research [11]. This project has captured the headers of actual network traffic from sites across the Internet. The traces used in this experiment were captured at the San Diego Supercomputing Center's Internet connections. The traces were collected during the weeks of July 19, 1999 and August 9, 1999. The header collection hardware produces perfectly accurate traces, but buffer limitations restrict the length of each trace to 90 seconds.

The network topology used for this simulation is shown in Figure 5. This structure was chosen so that the behavior of TCP under bridge congestion could be studied. There are no bottlenecks at either the TCP or trace generating nodes or their connecting links. The two traffic streams feeding into the single congested link result in congestion at the bridge, which implements simple tail-drop queueing. Each trace was simulated with the bandwidth of the congested link set to 20, 30, and 40Mbps to provide different levels of congestion.

Because of the short length of the traces, each observation consisted of 1/2 second to take the "SNMP" measurement (actually done with the simulator trace) and 1/2 second for the TCP connection, followed by a 1/2 second pause before the next observation began. Over 10000 observations were taken using the short data sets. For each tracefile, the time series models were trained for the first 8 to 16 observations, and the prediction quality measured on the remainder.

$A_t(\mathcal{N})$ was built using the series of times recorded for the 1/2 second TCP transfers.

The SNMP-based predictions were done by applying the time series models to the series of trace-based "SNMP" available bandwidth measurements. For each data set, $A_n()$ was created using the other data sets.

5.1. Heavily congested networks

The simulator allowed the analysis of performance prediction under much heavier traffic loads than our first experiment. As a link grows more congested, the available bandwidth reported by SNMP approaches zero. The available bandwidth, however, does not indicate the offered load, which is the amount of data applications are attempting to send through the link. For the same low available bandwidth measurement, the offered load may range from the link's bandwidth to orders of magnitude higher.

If the link is only lightly congested, a path for which SNMP reports little available bandwidth may actually provide an application with a higher rate than the amount available. This behavior can occur if the competing traffic's rate is reduced in response to the new application's traffic.

If the offered load is significantly higher than the link's bandwidth, then it will be hard to get any data through. The router preceding the congested link will be dropping many packets already, and the competing traffic will be just as quick as the new application to use any bandwidth that becomes available.

These behaviors should be represented in $A_n()$ and are dependent on the type of network and competing traffic involved. In these cases, additional information may be obtained through the count of dropped packets, which is also available through SNMP. However, it is doubtful that such a link would be used for any performance-sensitive distributed applications, therefore we have not examined the usefulness of incorporating this information into the model.

5.2. Results

Our initial analysis of the data revealed an interesting effect of the heavy congestion generated by some traces. Although the level of congestion produced by

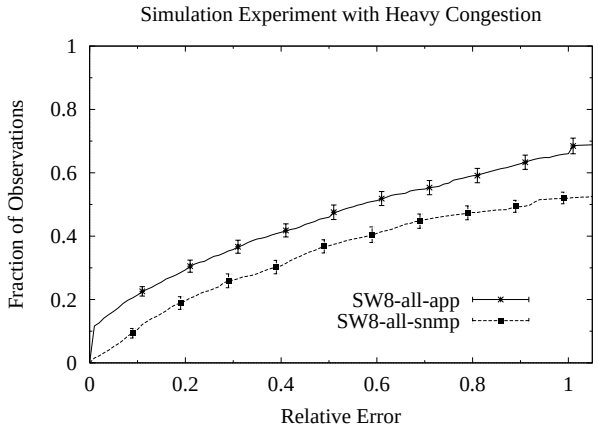

Figure 6. Relative error distribution with 95% confidence intervals for the simulation experiment including observations during extremely heavy congestion.

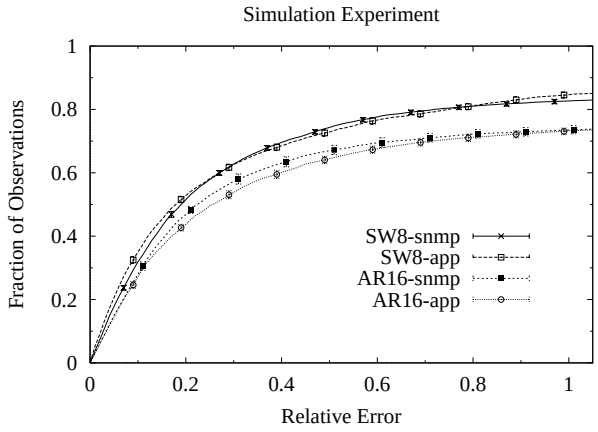

Figure 7. Cumulative relative error distributions with 95% confidence intervals for application- and network-based prediction in the simulation experiment. Both 16th-order autoregressive (AR16) and 8 observation sliding window mean (SW8) predictions are shown.

the simulation was sometimes unrealistic, because the traffic in the traces would have adapted to the higher congestion level, heavy congestion does occur in real networks. The data shown in Figure 6 was surprising because there is a significant difference between the two prediction techniques and because the accuracy of both techniques is less than seen in most other experiments.

Studying the data revealed that the majority of the errors occurred when the network was under extremely heavy congestion and that the network-based technique was more prone to these errors than the application-based technique. This discrepancy is believed to be due to the network-based technique's inability to measure the offered load on a congested link, whereas the application-based technique provides a history indicating whether the link is only marginally congested or seriously overloaded. Because these errors only appeared under extreme congestion and because such scenarios are of little interest to most applications, observations where less than 1% of the link's bandwidth was available were removed from the data used in Figure 7.

Figure 7 shows the aggregate results from all of the simulations. The simulation results also confirm that the accuracies of the application- and network-based techniques are very similar. It is interesting to note that in the simulation results, the results group according to the time series model chosen, whereas in the testbed results in Figure 4, the best results from each technique were with different time series models. This merely serves to illustrate that it is very difficult to select the most appropriate model. Prediction systems such as RPS [8] compute several time series models and report results from the one with the lowest error. Therefore, it is probably most appropriate to consider only the top curves from each technique.

6. Statistical metrics

The similarity in the results from the two techniques is very promising, because it indicates that a more efficient technique offers the same accuracy as the established technique for measuring network performance. The confidence intervals in Figures 4 and 7 indicate that both techniques have similar variability, but the differences between the testbed and simulation results require some explanation.

The results from the testbed experiment were divided into separate independent data sets. Because the background traffic was changing continuously, each data set was taken under different network conditions. This accounts for much of the variability—it is much easier to make accurate predictions on a lightly congested network versus a heavily congested network. For the simulation results, however, the runs produced using the individual tracefiles were too short to produce enough data for a CDF. Instead, the distributions were created by randomly dividing the data from all simulation runs amongst 30 sets. This randomization homogenized the data and removed much of the variability from the final result.

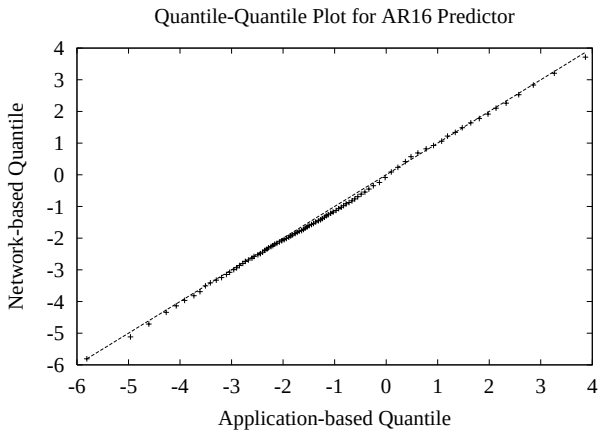

Figure 8. Quantile-quantile plot of the CDFs for each prediction technique. This is a plot of the logarithms of the values so that the similarity of the two techniques in both the low and high relative error regions can be seen. The $y = x$ line is shown for reference.

It is very important to consider the differences in predictability as network conditions change, as seen in Figure 4. Understanding and reporting to the application the accuracy of a prediction made under particular network conditions is just as important as making the prediction itself.

The CDF plots of the results tend to obscure the results for both low and high relative errors, in the first case because the distribution rises quickly and in the second because the distribution is long-tailed. To compare the two techniques over the entire distribution, Figure 8 presents a quantile-quantile plot for the simulation experiment. The linearity of the plot indicates that the distributions are nearly identical.

7. Topology discovery

Knowledge of network topology is essential for application mapping because the links that are shared by different components of the same application (internal sharing) have a large effect on the performance of that application. Topology knowledge also simplifies scheduling algorithms because it is possible to schedule in a hierarchical fashion, rather than analyzing all combinations of machines. Finally, it is impossible to use network-based techniques without first learning the topology to determine which components are involved in the path about which queries are being made.

Topology discovery is difficult because user-transparency has been a great driving force behind the success

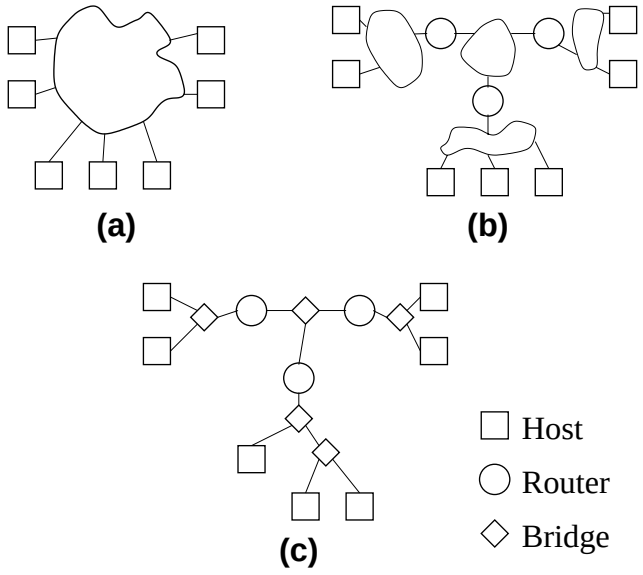

Figure 9. A view of networking at different levels of detail. (a) The view presented to the user. (b) The view at the IP routing layer, where each host and router explicitly forwards packets to the next component in the path. (c) The view including Ethernet bridges, where each bridge learns where the hosts and routers are and transparently forwards the packets towards their destinations.

of networking. As a result, there are no good protocols for determining topology. However, the necessary information can be extracted using SNMP with enough perseverance.

7.1. Network structure

Figure 9(a) shows the network view that is presented to the user and that is preserved by most programming libraries. In Figure 9(b), the IP routers connecting these machines are exposed. These are the easiest components to detect. In fact, the `traceroute` program can be used to detect routers between hosts.

The second level of transparency is exposed in Figure 9(c). Here, the bridges that form the Ethernet LANs connecting the machines are exposed. This is the most difficult level of topology to penetrate, although it is the most common LAN infrastructure. The difficulty comes from the beauty of the transparent bridging protocol. The algorithms that the bridges use to determine how to form the LAN and how to forward packets require no global knowledge, nor do the hosts talk directly to the bridges [14]. Thus, the goal of transparency is completely met, at the expense of the ease of determining the topology from the bridges. It should be noted that modern networks are typically built with

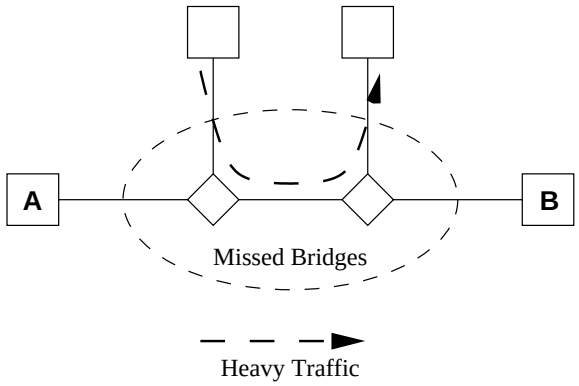

Figure 10. An example of a network configuration where missing bridges can produce misleading results. In this case, missing the two bridges between A and B misses the congested network link between them and may vastly overestimate the available bandwidth.

"switches," which are essentially bridges with many ports.

Despite the difficulties, it is necessary to locate all components of a network before using network-based queries on that network. Figure 10 shows an example where available bandwidth predictions will be useless because a congested link occurs between two undiscovered bridges.

7.2. IP routing

IP routing topology is easy to determine because each host and router uses a routing table that lists the next hop on the route used to reach each destination. SNMP can obtain this table, which makes determining IP routing topology a simple matter of following the routers hop-to-hop from source to destination.

7.3. Bridged Ethernet

The Ethernet bridging algorithm is much more complex than the IP routing algorithm. A bridge learns where to forward packets by listening to all traffic on the links to which it is attached. Whenever it sees a packet, it stores its source address and the link it was received on. This information forms the forwarding database used when forwarding packets to their destination. When the bridge receives a packet from a destination not listed in its forwarding database, it "floods" this packet on all of its ports. When the unknown machine responds to this packet, the bridge adds it to its database. This algorithm is known as transparent bridging, which is currently used almost exclusively on Ethernet LANs. More information can be found in Perlman's book [14].

Because this algorithm is completely transparent to the hosts, it is difficult to find bridges automatically. One solution is to obtain a list of bridges from an external source, such as the local network manager. Secondly, because bridges only learn a host's location when they receive a packet from that host, care must be taken to ensure that the forwarding entries are present in the bridge's database.

This situation motivates a rather complex algorithm for determining the bridging topology. Before beginning, the routing topology must be determined, as in Figure 9(b). Once that has been accomplished, the bridging topology, as found within each cloud in that diagram, can be determined.

The algorithm begins with a set of endpoints, E, consisting of all of the hosts on the Ethernet for which the topology is desired, as well as the routers used to connect this Ethernet to other networks. Also known is the set of bridges, B, used in this network. Bridging topology is defined to be a tree, with the members of E forming the leaves and the members of B forming the internal vertices. The basic approach to determining this topology is to go through the members of B, querying for the ports to which they forward packets routed to members of $E \cup B$. This information tells us the edge of each vertex that is used to reach every other vertex. This knowledge is sufficient to construct the complete topology of the tree.

The difficulty of this algorithm is not in deriving the topology from the bridges' forwarding databases. Rather, it is in ensuring that the needed entries exist in the forwarding database. A two phase approach is used to obtain this information with reasonable efficiency.

Because there are $O(|B|(|E|+|B|))$ queries to make, and implementation difficulties make each query time consuming, it is important to reduce the number of queries whenever possible. The goal of the first phase is to determine the set of nodes, $B_u \subseteq B$, that are used in the network topology connecting E. This is done by querying each member of B for its forwarding port for each member of E. If the bridge uses the same port to reach all members of E, then the bridge cannot be part of the topology. If a bridge forwards packets to E using different ports, however, then it is needed to form the

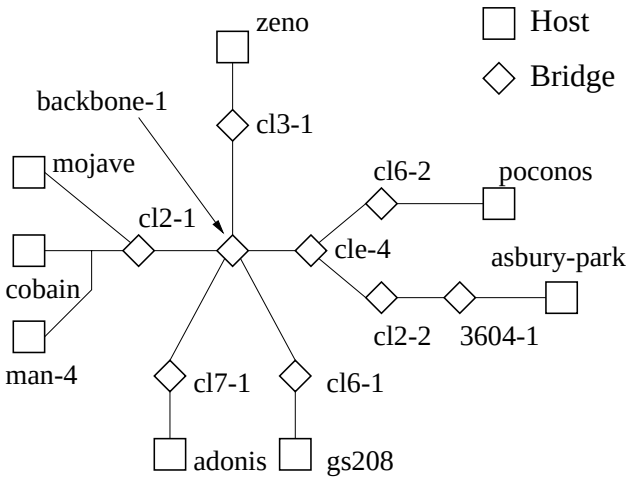

Figure 11. Topology determined from the CMU Computer Science Department's network. The process was begun with the hosts seen here and the 44 bridges used in our department. Note that cobain and man-4 were found to share a single Ethernet through a hub. The actual department network is much more complex—the algorithm prunes the graph to include only those bridges used to connect the set of hosts being used.

interconnection topology.

Because bridges learn passively, each bridge in B must have seen a packet from each member of E to have an entry in its forwarding database for that host. To ensure that this table is complete, all members of E periodically ping all other members of E before and during the data collection. This guarantees that if a bridge is on the topology between any two members of E, it will have seen packets from both members and will have their entries in its forwarding database. Note that it is generally not possible for users to have routers send pings, but routers do respond to pings, so if all hosts are sending pings to a router, the router's replies to the pings will ensure that its entry is present in the forwarding databases of the bridges.

Once B_u has been determined, the second phase of this algorithm begins by expanding the list of machines being pinged by the hosts to $E \cup B_u$. This forces the bridges to learn about each other's location, information not normally needed for transparent bridging. Finally, all members of B_u are queried for where they are forwarding packets to all other members of B_u.

Following the completion of this algorithm, the ports used by each bridge to forward packets to $E \cup B_u$ are known. Because the bridging topology is defined to use a tree, it is easy to extend this information to complete the topology.

Figure 11 shows the bridging topology between several machines at CMU. This structure was determined using the above algorithm, beginning with a list of the 44 bridges used in our departmental LAN. The correctness of this topology was verified by our network manager after the algorithm was run.

The $O(|B|(|B| + |E|))$ complexity of this algorithm and the high cost of each access to a bridge's forwarding database make this algorithm impractical for runtime use. For example, the topology in Figure 11 took approximately 30 minutes to discover. Fortunately, typical Ethernet LANs do not change topology often, and it is easier to verify that the topology is still accurate than to discover it. For these reasons, the bridging topology should be stored in a database where applications and prediction systems can make use of it. Periodic verification and rediscovery can be used to keep the database current.

8. Practical considerations

Our research has demonstrated that SNMP, already supported by almost all of the current networking infrastructure, is sufficient for obtaining the information needed to determine and predict performance directly from the network. Although it is not an ideal interface for this purpose, it allows the network-based approach to performance prediction to be explored and utilized on existing networks. Demonstrating the value of this approach by using it in real systems and applications should result in the development of more appropriate interfaces for network components. However, both administrative and technical considerations must be addressed to provide a better interface for performance prediction purposes.

The administrative complication is primarily accessibility. Typically, SNMP access is only allowed from machines on the local network, and it is usually impossible to make SNMP queries to network components on an ISP's network. Security and privacy are the two primary reasons for this. Security is actually a technical concern; because the designers of SNMP were unable to agree on a workable security protocol, there is little security in current implementations, therefore a minimal security level is achieved by restricting access to local hosts. ISP's are generally concerned about privacy, not wishing to divulge information about the con-

gestion levels of their services. Furthermore, because SNMP queries can be expensive, no one wants to open their network up to excessive load or even denial-of-service attacks with SNMP. We are currently pursuing combining network-based data with benchmark-based data to provide predictions in environments where direct network queries are only available for portions of the network.

Although RFCs describe the behavior of SNMP implementations, the standards and their implementations have not resulted in consistent interfaces between different manufacturers. For instance, the forwarding databases in Ethernet bridges are particularly troublesome. Some allow queries to be made for the forwarding port of a specific address. Other implementations are designed only for traversal, requiring the same query to be reformulated as a query for the subsequent entry from the numerically preceding address. Furthermore, some bridges remove the forwarding database if queries are made to it too rapidly, apparently as a security measure.

Finally, although SNMP provides much of the information needed for distributed computing, it is difficult to get it in the form required. For example, there are traffic counters for each port, but determining a traffic rate requires multiple, carefully timed queries. It would be much more appropriate to have the router calculate its own time-averaged rate. Preliminary work toward this goal is discussed in the APMMON Internet-Draft [7].

9. Related work

A variety of systems exist for providing network status information. NWS [21] and Prophet [19] provide applications with benchmark-based predictions. SPAND [17] records similar data by storing applications' actual performance during execution and making this data available to help future applications.

All prediction systems described here utilize time series prediction techniques [3]. Wolski has studied several different time series models for their usefulness in predicting network performance [20].

SNMP has much broader uses than those that are described here. It is used to control and monitor a wide range of network resource properties [18]. Busby has explored using SNMP to gather information about both network and CPU resource as an addition to NWS [4]. The techniques evaluated in this paper for SNMP-based bandwidth measurement were developed for the Remos system [9].

Recent network research has focused on modeling the self-similarity in network traffic, and these models may lead to more realistic traffic than Poisson processes. We used fractional Gaussian noise to generate self-similar traffic for our experiments [12]. More realistic wavelet models have been investigated more recently [15], and we are examining using them on our testbed to generate synthetic traces modeled after actual network packet traces.

10. Conclusions

Tools for discovering network performance and topology are an important part of the support infrastructure for distributed computing. This paper demonstrates that the network-based approach has significant advantages over the benchmark-based approach for scaling and topology discovery, without sacrificing accuracy. Importantly, this approach can be implemented today, using the SNMP structure already included in networking hardware for the purpose of network management. The benefits of the network-based approach outweigh the inherent limitations of using low-level information to predict end-to-end performance.

Major challenges remain for the distributed computing community to ensure the availability of the information needed for network-based predictions. Network administrators and companies must be encouraged to provide easier access to the performance data needed to properly schedule applications. Furthermore, working with the networking community may lead towards the incorporation of more beneficial and convenient features for performance measurement into SNMP and network devices.

The techniques described here are being implemented in the Remos system [9]. Remos is available from our website,
`http://www.cs.cmu.edu/Groups/CMCL/remulac`.

Acknowledgements

Effort sponsored by the Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Com-

mand, USAF, under agreement number F30602-96-1-0287. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

References

[1] S. Bajaj et al. Improving simulation for network research. Technical Report 99-702b, USC Computer Science Department, September 1999.

[2] J. Bolliger and T. Gross. A framework-based approach to the development of network-aware applications. *IEEE Transactions on Software Engineering*, 24(5):376–90, May 1998.

[3] G. E. P. Box, G. M. Jenkins, and G. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, 3rd edition, 1994.

[4] R. Busby, M. Neilsen, and D. Andresen. Enhancing NWS for use in an SNMP managed internetwork. In *Proceedings of the 2000 International Parallel and Distributed Processing Symposium (IPDPS'00)*, May 2000.

[5] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Structure of management information for version 2 of the simple network management protocol (SNMPv2). RFC1902, January 1996.

[6] E. Decker, P. Langille, A. Rijsinghani, and K. McCloghrie. Definitions of managed objects for bridges. RFC1493, July 1993.

[7] R. Dietz. Remote monitoring mib extensions for application performance metrics. IETF Internet-Draft, July 1999. Work in progress.

[8] P. A. Dinda and D. R. O'Hallaron. An evaluation of linear models for host load prediction. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, August 1999.

[9] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A resource monitoring system for network-aware applications. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 189–196. IEEE, July 1998.

[10] K. McCloghrie and M. Rose. Management information base for network management of TCP/IP-based internets: MIB-II. RFC1213, March 1991.

[11] National Laboratory for Network Analysis (NLANR). Passive monitoring and analysis via packet header traces. http://moat.nlanr.net/. National Science Foundation Cooperative Agreement No. ANI-9807479.

[12] V. Paxson. Fast, approximate synthesis of fractional gaussian noise for generating self-similar network traffic. Technical Report LBL-36750, Lawrence Berkeley National Laboratory, April 1995.

[13] V. Paxson and S. Floyd. Why we don't know how to simulate the internet. In *Proceedings of the 1997 Winter Simulation Conference*, pages 1037–44, 1997.

[14] R. Perlman. *Interconnections: Bridges and Routers*. Addison-Wesley, 1992.

[15] V. J. Ribeiro, R. H. Riedi, M. S. Crouse, and R. G. Baraniuk. Simulation of nongaussian long-range-dependent traffic using wavelets. In *Proceedings of ACM SIGMETRICS '99*, pages 1–12. ACM, May 1999.

[16] J. Saltzer, D. Reed, and D. Clark. End-to-end arguments in system design. *ACM Transactions in Computer Systems*, 2(4):277–288, November 1984.

[17] S. Seshan, M. Stemm, and R. H. Katz. SPAND: Shared passing network performance discovery. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 135–46, December 1997.

[18] W. Stallings. *SNMP, SNMPv2, and RMON*. Addison-Wesley, 2nd edition, 1996.

[19] J. B. Weissman and X. Zhao. Scheduling parallel applications in distributed networks. *Cluster Computing*, 1(1):95–108, May 1998.

[20] R. Wolski. Dynamically forecasting network performance using the network weather service. Technical Report CS-96-494, UCSD, 1996.

[21] R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High Performance Distributed Computing Conference (HPDC)*, pages 316–25, August 1997.