A Measurement Methodology for Wide Area Internets

Calton Pu, Frederick Korz, and *Robert C. Lehman*[1]
Department of Computer Science
Columbia University
New York, NY 10027

Technical Report No. CUCS-044-90
Revised – March 1991

Abstract

Measuring the performance of applications running over live wide area internets (WAINs) is important and difficult. Directly observed performance and availability can help application developers and network managers. To compensate for the difficulty of controlling important variables such as message path and network load, we developed the Layered Refinement (LR) measurement methodology. LR tools use generic user-level software to collect data simultaneously at important software layers. LR data reduction techniques reveal interesting phenomena on network routing, latency, and availability, in addition to application response-time and RPC performance over the Internet. We illustrate the application of LR with a series of measurements and some of the phenomena observed.

[1] Current address: IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598

Contents

1 Introduction

Despite the tremendous growth of wide area internets (WAINs) and the number of users they connect, we find little, if any, published data on actually *measured* end-to-end performance of WAIN application software. Measured data is important to network managers, distributed application developers, and network modelers because traditional techniques such as analytical modeling, simulation, and testbeds do not capture the complexity of a large WAIN. Our experience in measuring distributed applications running over the Internet since 1987 shows some of the difficulties and the methodology we developed to deal with them.

The foremost difficulty in measuring the performance of distributed applications running over WAINs is the difficulty in controlling the environment. Another serious problem is the constant evolution of WAINs. For example, Columbia switched from a direct ARPANET connection (56 Kbaud) to an NSFNet connection (through NYSERNet, T1 lines at 1.5 Mbit/sec) in 1988. In addition, the operating systems (OS), the applications, and router/gateway software have had several new releases during the period of our investigation. A third problem happens during run-time: in a packet-switched network such as the Internet each packet can be routed differently, causing potentially large variances in measured data. Messages travel through many intermediate gateways, routers, and shared media, each with its own queueing delays. Consequently, the raw data (for example, the points in Figure 6 of Section 3) shows an apparent randomness.

We developed the *Layered Refinement* methodology (LR) to measure distributed application performance characteristics such as response-time, effective bandwidth and availability on a live Internet (in contrast to controlled laboratory experiments). LR consists of three parts: (1) divide the application into layers and design measurements for each layer; (2) collect data simultaneously on all the important layers; (3) iterate as many times as necessary the process of analyzing and refining the measured data to improve precision. With LR we have successfully measured and analyzed the performance of two distributed applications running over the Internet. Although the numbers we obtained are fundamentally different from laboratory experiments, they can be reproduced with high confidence. Finally, our methodology uses only simple software measurement tools, making it very practical and widely applicable. The main contribution described in this paper is the formulation and demonstration of the LR methodology.

This paper reports on the results of our measurements, data collection tools, and data analysis methodology, all of which may be useful for both developers and users of distributed applications over WAINs. We describe the experiments in Section 2. The LR methodology is summarized in Section 3. Sections 4 and 5 contain the analyses of data on WAIN and application performance, respectively. Related works, such as kinds of packets and their size, are summarized in Section 6. Section 7 concludes the paper.

2 Layered Experiments

Our goal in developing the LR methodology is to measure many distributed applications, without having to completely redesign the data acquisition and data reduction toolkit. The technical details of the experiments are omitted since their purpose is to illustrate the phenomena observed by the LR methodology. Analysis of Camelot transactions will be described in another paper [13].

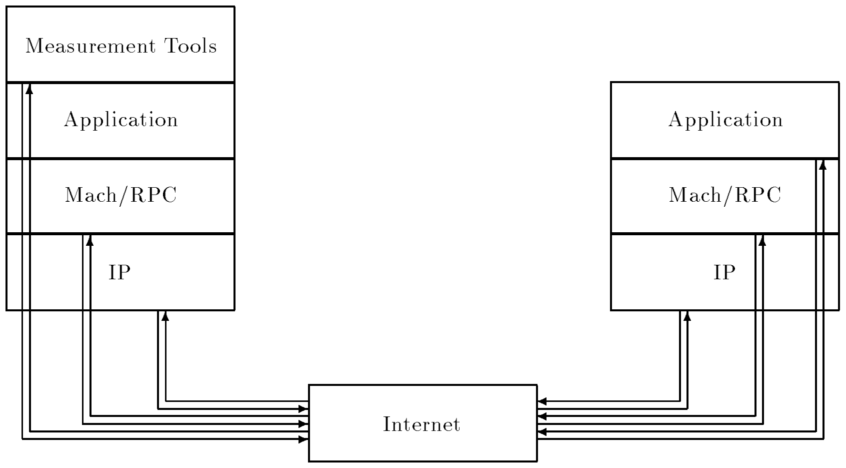


Figure 1: The Application Model

2.1 Layered Model

Our model of an application software package running over WAIN is shown in Figure 1. The network connects the two ends of the application, which may or may not be symmetric. We call the network box the horizontal dimension of our model, since messages flow through the network within the communications protocol system software. At the end nodes, we call the multiple layers of software the vertical dimension of our model, since the messages flow up and down from applications into the system software and back.

We are interested in measuring the end-to-end performance of application software, through both the vertical and horizontal dimensions of the model. The measured numbers can give us a better understanding of WAIN applications, and to help us find their bottlenecks in need of optimization. This multi-layer model divides the application and the OS into smaller chunks for detailed performance analysis.

The first observation about the model is that we should make each layer in the vertical and horizontal dimensions significantly thick for the measurements. In other words, the amount of time a message spends in each layer should be of approximately the same magnitude. Otherwise, we subdivide the dominant layer (in cost) appropriately. This requirement is system measurement common sense. It is not specific to WAIN measurements. However, we note that relatively thick layers make WAIN measurements more tolerant of the coarse granularity data produced by generic software tools.

The second observation is that each layer in the model should present an external interface to the user. This means that we should be able to write a program to access the facilities of each layer, independent of the layer above. Although this requirement is not made explicit in LAN measurements (Section 6.2), those researchers had full access to the source code of their systems. In other words, they instrumented their system software for measurement purposes. LR does not require modifications to each layer of system software, just that each layer be independently accessible to users.

2.2 Client/Server Experiment

Our experimental environment, the Internet, is a WAIN with more than 100,000 nodes spanning the world [11]. The Internet is organized hierarchically, with high-speed backbones in the center and slower links towards the outer fringes of the graph. The backbone in the United States is the NSFNet. A number of regional networks connect into NSFNet, usually through NSF supercomputer centers. For example, Columbia's connection to the Internet is a T1 link into NYSERNet, with gateway to NSFNet at the Cornell Supercomputer Center.

Our first data set was obtained by measuring, from Columbia, the round-trip time of the Webster's Ninth New Collegiate Dictionary Server running at the University of Washington. The hardware on both ends was a NeXT machine running the Mach operating system [1]. This data, from now on called the *Webster sample*, was obtained during a seven-day period in September of 1990 and is used primarily to show the different routes packets have traveled.

On the Washington side, the Webster Server receives a word in the request, looks up the word in the dictionary and returns the first N characters of the result, where N is a function of the request message size. We truncate the reply to N characters to control the number and size of packets transmitted. On the Columbia side, the dictionary client doubles as the measurement tool. The client makes a series of requests for lookups from a list of words. Each request is timed and spaced by a one-second delay. After the client finishes the iterations it records a short performance summary. In the Webster sample, each batch contains 11 iterations and runs every half an hour.

2.3 Distributed Transaction Experiment

Our experiments started in 1987 as an effort to measure the performance of the Camelot distributed transaction facility [18] over the Internet. Camelot distributed transactions consist of several RPCs and processing at both ends. A standard statistics-gathering program, called the Camelot Performance Analyzer (CPA), is used to obtain round-trip times between Columbia and CMU. The CPA digests and records the raw data.

Camelot measurements have been made between several machines at Columbia (two Microvaxes) and CMU (primarily a Microvax and an IBM/RT), all running Mach. There are several similarities as well as differences between the Camelot experiments and Webster experiments. Both applications run on Mach OS, using its RPC. Webster has only one round trip while Camelot may have several when multiple servers are involved in a two-phase commit protocol.

In this paper, we use the results from a series of measurements made from November 1990 to January 1991, called the *Camelot sample*. The Camelot sample has in each batch 21 CPA points, plus 11 RPC and `ping` iterations every half an hour. We ran batches of both read transactions and write transactions.

3 Layered Refinement Methodology

The methodology described in this section has been developed through several years of experience. The challenge we would like to meet is to design an extensible methodology, capable of adapting to network bandwidth upgrade of two order-of-magnitude, new applications and network protocols.

3.1 Motivation

In measuring system performance, the best numbers are obtained from hardware-assisted measurement tools that do not introduce any distortion into the measured numbers. However, such tools are impractical over WAIN due to the expenses and the difficulties of installing them on the intermediate routers.

The next best alternative would be software-assisted measurements taken directly by each layer of software involved. Network administrative domains such as NYSERNet and NSFNet that implement the Simple Network Management Protocol [3] collect some information for each domain, but it is local, not generally available, and when available not publishable. The alternative is for each layer of the system to add some identification and timestamps to messages as they travel through the system. Three important problems make this approach impractical. First, such services are often simply not implemented in existing operating systems and network protocols. One example is the IP Record Route option, which would record the node-ids through which the message travels. But unfortunately the Record Route option is not implemented in most routers. Even if it were available, the diameter of the Internet (about 20 currently) and the typical distance (the Camelot and Webster samples have from 12 to 15 hops) have already outstripped the planned capacity of Record Route (only 9 or 10 address slots).

The other two problems are the measurement cost and clock granularity. Taking timestamps to produce the time interval is expensive (usually requiring a kernel call) in most existing operating systems. The cost problem is compounded by the additional message size. A small message may become large due to the number of layers it traverses horizontally in the network and vertically within a node. The third problem is the coarse granularity of clocks in most workstations. For example, we have experimentally found a clock precision of 10 milliseconds on a Microvax II running Mach, which makes direct measurements of our applications infeasible (we are interested in operations on the order of 1 millisecond or less).

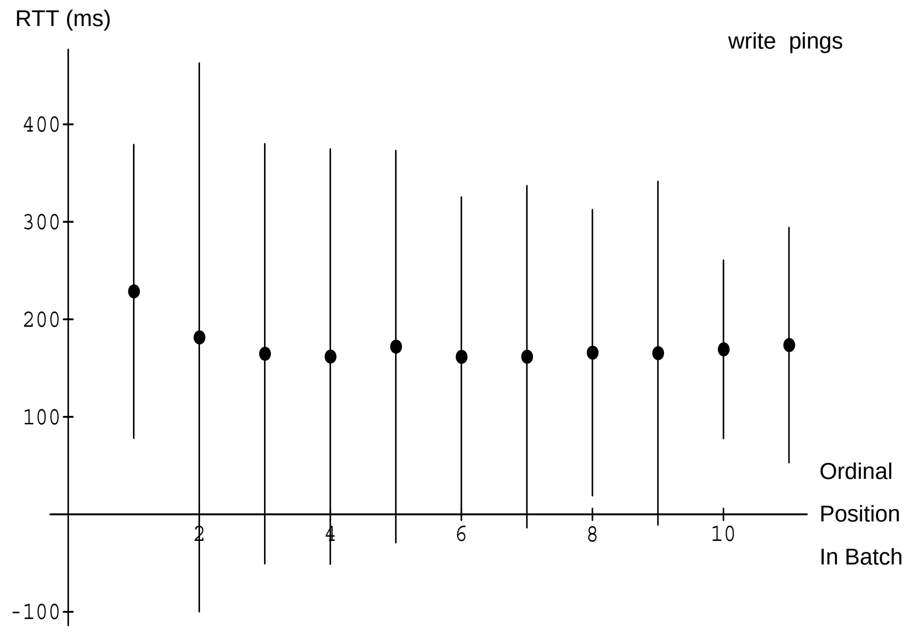
Given that we have little hope of using specialized hardware or software to make measurements over a live network such as the Internet, we had to develop a measurement methodology using only generic software tools. These tools have to be very simple, since they cannot modify the underlying network system in any way. To filter out the noise present in an uncontrolled WAIN, we collect a large amount of data and use non-trivial data reduction techniques, which serve the same function as laboratory controls that isolate the phenomena of interest.

3.2 Data Reduction at Each Layer

Warm-up effect: Both the Camelot[2] and Webster samples take 11 points in each batch. Figure 2 shows the mean response-time of each point by their position in the batch for the Camelot sample. The figure shows a systematic bias for the first point of each batch to take longer time. This is known as the "warm-up" effect, due to start-up costs such as program initialization, virtual memory loading, cache filling, and host name to IP address translation. Thus we drop the first point of each batch in some of data analysis to eliminate the systematic bias introduced by warm-up.

Distribution: To show the difficulties of modeling the measured data, we plot the frequency distribution of observed round-trip times. Figure 3 shows a relatively simple curve.

The sole exception is CPA for which 21 points were taken in each batch.



Camelot sample, ping layer, individual points, no filtering.

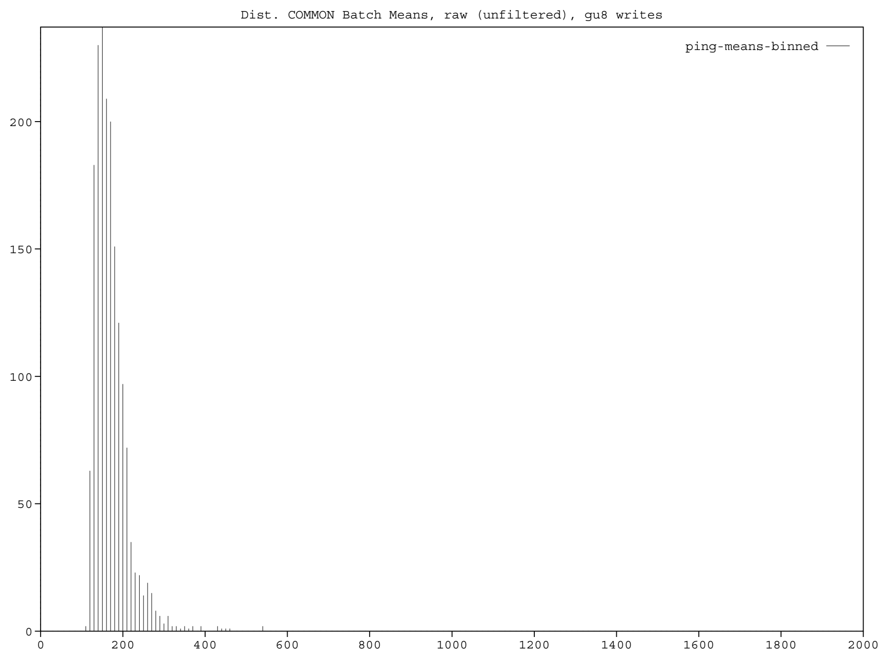
Figure 2: Round-Trip Time (mean and std. dev.) by Position

The X-axis is round-trip time measured by `ping` and the Y-axis shows the number of occasions a given time was observed. The asymmetric unimodal curve in Figure 3 is typical of low-level protocols (see another WAIN measurement study [8]). However, complex applications such as Camelot transactions are different. Figure 4 shows a bimodal distribution of Camelot write transactions measured at the same time as the ping numbers in Figure 3. The bimodal distribution (with first mode at about 500 ms and second at about 1500 ms) is consistent in the Camelot experiments. (The spike at 1910 in Figure 4 simply accumulates all the data points higher than 1910.)

Both figures show data points concentrated near the left side around their mode, with a right-side tail towards long response-times. The points around mode are considered “normal”, presumed to have followed the usual path. A response-time significantly longer than the mode can be due to many different phenomena in the network, such as lost packets, retransmission, and delays in some router. To improve the signal/noise ratio of our data, we filter out the “abnormal” points.

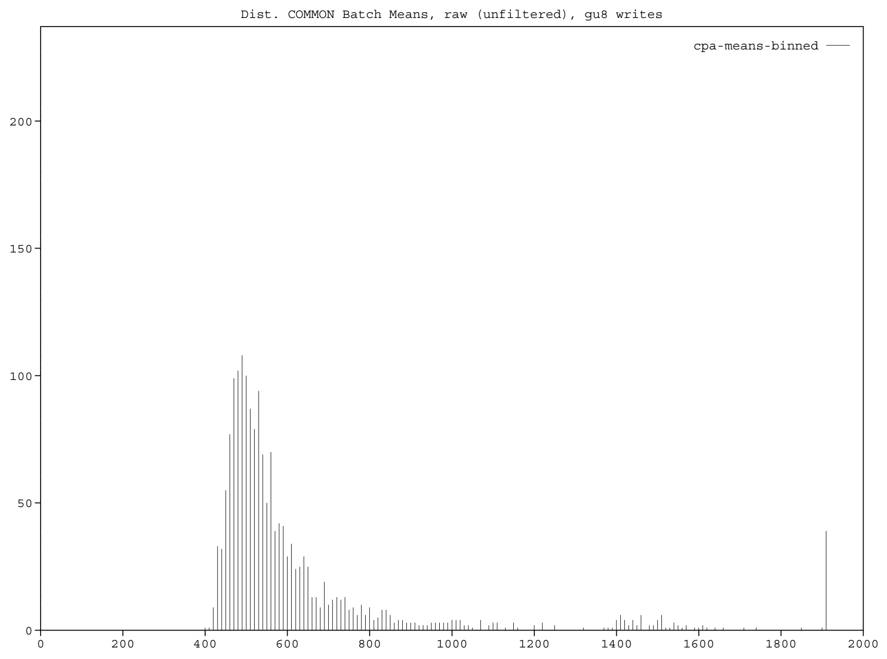
Outliers: Table 1 illustrates our filtering of outlier points. The raw data used is from the Camelot write sample, shown also in Figure 6. Column 1 describes the kind of data analyzed in that row. Column 2 shows the total number of batches (each containing 10 or 11 points) accumulated. Then we have the mean, mode, and median of the distribution of accumulated mean values from the batches. Column 6 shows the value of a batch at the 95% cut-off point, and column 7 the value of *upper fence*, defined as 1.5 times the inter-quartile range (distance between 25% and 75%) greater than the 75% cut-off point. Upper fence (page 43 in [20]) is a commonly accepted definition of outliers.

Table 1 shows that mode is consistently smaller than median, which is smaller than mean.



Camelot sample, ping layer, means of batches.

Figure 3: Distribution of Data Points Accumulated (ping)



Camelot sample, CPA layer, means of batches.

Figure 4: Distribution of Data Points Accumulated (CPA)

Data Type	Number of Batches	Mean of Means	Mode of Means	Median of Means	95% of points	Upper Fence
Raw Data	1735	171 ms	150 ms	162 ms	249 ms	254 ms
Drop First	1735	165 ms	140 ms	155 ms	239 ms	241 ms
95% of Raw	1732	153 ms	140 ms	146 ms	205 ms	213 ms
Upper Fence of Raw Data	1724	143 ms	130 ms	139 ms	183 ms	183 ms

Camelot sample, ping layer, comparison of different filtering threasholds.

Table 1: Refining `ping` Data

Data Filtering	RPC – Ping				CPA – RPC			
	mean	mode	median	correlation	mean	mode	median	correlation
Raw Data	136 ms	40 ms	47ms	0.461	435 ms	270 ms	316 ms	0.369
Drop First	143 ms	40 ms	50 ms	0.474	383 ms	270 ms	289 ms	0.413
95% percentile	46 ms	40 ms	45 ms	0.525	293 ms	290 ms	286 ms	0.485
upper fence	49 ms	40 ms	47 ms	0.601	283 ms	270 ms	278 ms	0.487

Camelot sample, ping, RPC, and CPA layers, filtered and correlated.

Table 2: Spearman Correlation of Multi-Layered Data

It also shows a considerable reduction in the value of mean after filtering the outliers (rows 3 and 4).

3.3 Simultaneous Multi-Layer Data Collection

Given the number of uncontrollable variables in the Internet, independent measurements of layers would give us little information about their relationship with each other. Therefore we measure all the layers concurrently. The idea is that whatever affects one layer would affect the other layers as well. Although we cannot establish causality between the numbers measured from different layers, we can *correlate* them. Using correlation we can separate the participation of low level software by subtracting the round-trip times of one layer from the layer above. For example, subtracting the `ping` cost from the RPC cost we can learn the pure RPC cost. This analysis is fundamental to the understanding of each software layer, especially of the higher layers, including the application. On top of `ping` we have RPC, measured with a tool called mirror-RPC, which simply echoes its input. On top of RPC we have the applications.

Since LR collect the data from different layers at the same time, we would expect a strong correlation between them. Indeed this is the case, and Table 2 illustrates this point by filtering the outliers in several ways. Since the distributions are not symmetric, we do not use the usual Pearson correlation, which assumes normal distribution. Instead, we use the Spearman correlation (page 507 in [12]) of two adjacent layers. Table 2 shows that we

Data Type	ping			Mirror RPC			CPA write		
	mode	median	inter-q. range	mode	median	inter-q. range	mode	median	inter-q. range
Camelot Sample	150 ms	162 ms	44 ms	200 ms	208 ms	59 ms	490 ms	534 ms	149 ms
Control Experiment	120 ms	132 ms	33 ms	140 ms	145 ms	29 ms	440 ms	471 ms	91 ms
Presumed Interference	+30 ms	+30 ms	+11 ms	+60 ms	+63 ms	+30 ms	+50 ms	+63 ms	+58 ms

Camelot sample and Interference experiment, three layers, raw data.

Table 3: Self-Interference in Camelot Sample

have increasing correlation as the filtering techniques become more refined. Since we know the data sets are correlated by construction, this is a strong indication that outliers are due to isolated problems within the layer or in the network. Although the correlation increases monotonically with greater filtering, we can see some oscillation in the median and mode of the distributions. The mode change is due to the initial multi-modal distribution of the application (CPA). The filtering changes the median because each batch may be affected differently by the elimination of points, thus changing its mean.

3.4 Estimating Interference

Although simultaneous data acquisition allows us to correlate data, it also introduces potential interference between the several layers being measured at the same time. If packet processing were not the bottleneck in the system, then the interference would be small. However, in workstations the CPU overhead in network protocol processing is significant, as well as the application overhead. Therefore we designed experiments to quantify the amount of interference.

For the Camelot sample, we used separate experiments that sequentially run each layer, eliminating the competition between the layers. To check the correlation between the layers, we run a lower layer before and after a higher layer. The control experiment, therefore, consists of `traceroute`, `ping`, RPC, CPA, RPC, `ping`, and `traceroute`. Since the two runs of lower levels in each batch were always consistent, we only show one set of numbers for them in Figure 3. We see considerable interference for this case, since the difference between the Camelot sample and the control experiment is significant: 25% for `ping`, 40% for RPC, and 12% for CPA. The interference may be overstated, as the control experiment was run a few weeks after the Camelot sample. A careful design of future experiments will minimize the interference for resource-demanding applications.

3.5 LAN Control Experiments

To validate the LR methodology, we ran the same experiments over a LAN as control. The LAN experiments were done between two NeXT machines on the same Ethernet. However, they were not on an isolated network and were used as workstations. Therefore we have observed large variances in the means. Since we use the LAN experiments as control, we have used the minima as an approximation to laboratory measurements. Table 4 summarizes

Data Type	Mean of Minima	std.dev. of Minima	Remain. Data
`ping`	4.0 ms	0.2 ms	100%
mirror-RPC	19 ms	1.8 ms	100%
Webster	67 ms	31 ms	100%

LAN control experiment (replicating Webster sample)

Table 4: Summary of LAN Data (Minima)

LAN data obtained in four days in September 1990. We have applied the usual filters but no points were eliminated.

4 WAIN Performance

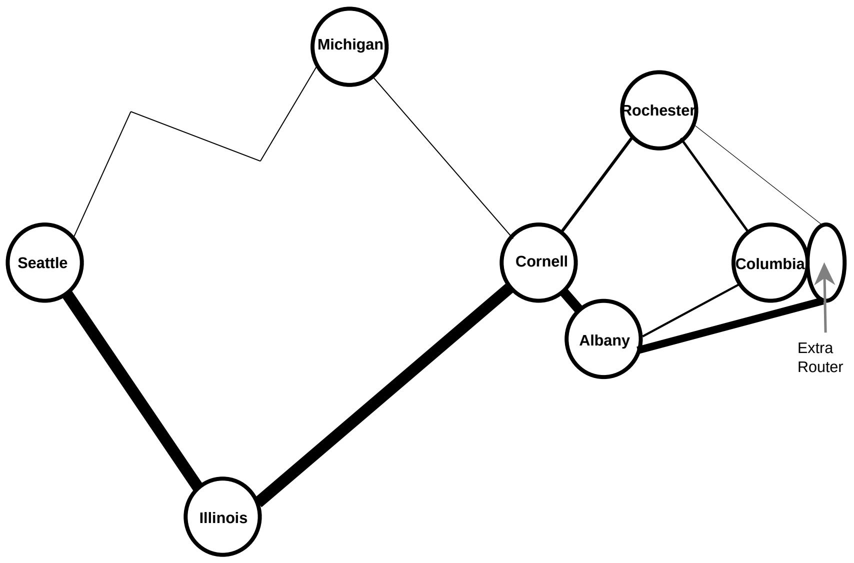
4.1 Internet Packet Routing

IP packets on the Internet travel from router to router until they reach their destination. The exact route is determined dynamically. If the packet route between the Measurement Tool and Application changes often, this will be a source of variance in round-trip time. In the experiments before October 1990, such as the Webster sample, routing seems stable for minutes or hours at a time. In the experiments after October 1990, such as the Camelot experiments, dynamic rerouting happens frequently, since different trials in the same batch (from a few seconds to a minute) may yield different routes.

Van Jacobson's `traceroute` program is the generic software tool we use to discover the current Internet routing. IP packets contain a time-to-live (TTL) field. At each router, IP decrements a packet's TTL by one, the packet is returned to the sender when its TTL reaches zero. `Traceroute` starts with TTL= 1 to find the nearest neighbor, and increments TTL gradually until the packets reach the destination. To check the correlation between the `traceroute` results and measurements of higher levels, we ran it before and after the main measurements. Only 1.5% of the Webster sample data showed different routes between the beginning and the end of measurements. Even for those rare cases, the routes were similar enough (switching between Rochester and Albany) that the rerouting would not have affected the timings.

Figure 5 shows the main routes observed by `traceroute` in the Webster sample (plus two additional days). The thickness of each edge represents the percentage of time a route is taken. The thickest line is between Cornell University and NSFNet's Seattle gateway through University of Illinois at Urbana-Champaign, representing about 92% of packets. The length of the edges represents a typical packet travel time between the gateways. For example, it takes about 20ms from Columbia to either Rochester or Albany, and about 50ms from Cornell to Illinois. The only exception is the link between Michigan and Seattle, which goes through Salt Lake City and California; it takes more than 100ms and is drawn as a jagged line, out of scale.

Each of the nodes in Figure 5 includes several local routers or gateways. We do not show the individual local routers since the travel time between them is very short. However,



Webster sample, `traceroute` layer.

Figure 5: Most Used Routes Columbia—Washington

in the figure we include a router on the Columbia end that was removed by NYSERNet personnel during our data collection period. Unfortunately this occurred at the end of the data collection, so we do not have enough data for a direct comparison between the travel time before and after the change. This is an example of permanent route changes that may occur in the Internet, introducing systematic bias into the measurement data. Another imminent change is the upgrade of the NSFNet backbone to 45 Mbits per second T3 lines.

Some of route changes are transient and there are several reasons a message may take a different route. First, IP congestion control may redirect packets around one or more busy routers. Second, routers may temporarily malfunction or crash, causing short or long delays until the routing tables are updated. One router bug exposed during the Webster sample created routing loops that lasted for a few seconds. Third, routers may be shut down for administrative purposes. For example, during the Camelot sample, a runaway machine forced the isolation of NYSERNet from the NSFNet backbone for significant periods of time [14].

Table 5 starts with Webster sample data filtered with a 1-second threshold (this is higher than the upper fence) as the universe set. To separate the routes, we use the `traceroute` data to determine when the messages travel via the same route. (Figure 5 shows another way of compiling that data.) The experiments sharing the same route are grouped together. The route through Albany, Cornell, and Illinois is the most popular with 56.5% of messages, followed by Rochester, Cornell and Illinois with 13.8% and the shortened (by one local router) route through Albany, Cornell, and Illinois. All the percentages are relative to the universe set. In 95% of the cases, separating the routes shows from 25% to 30% reduction in the magnitude of standard deviation.

Data Type	Mean of Means	std.dev. of Means	% Data Used
Raw Data	286 ms	765 ms	112 %
Thres. 1 sec	199 ms	45 ms	100%
Albany/Cornell/Illinois	186 ms	31 ms	79.3%
Rochester/Cornell/Illinois	222 ms	80 ms	4.7%
new Rochester/Cornell/Illinois	227 ms	32 ms	5.0%
new Albany/Cornell/Illinois	200 ms	31 ms	5.3%

Webster sample, ping layer, mean of batches, comparison of filtering thresholds.

Table 5: Separated Routes Decrease Standard Deviation

4.2 Protocol Response-Time and Bandwidth

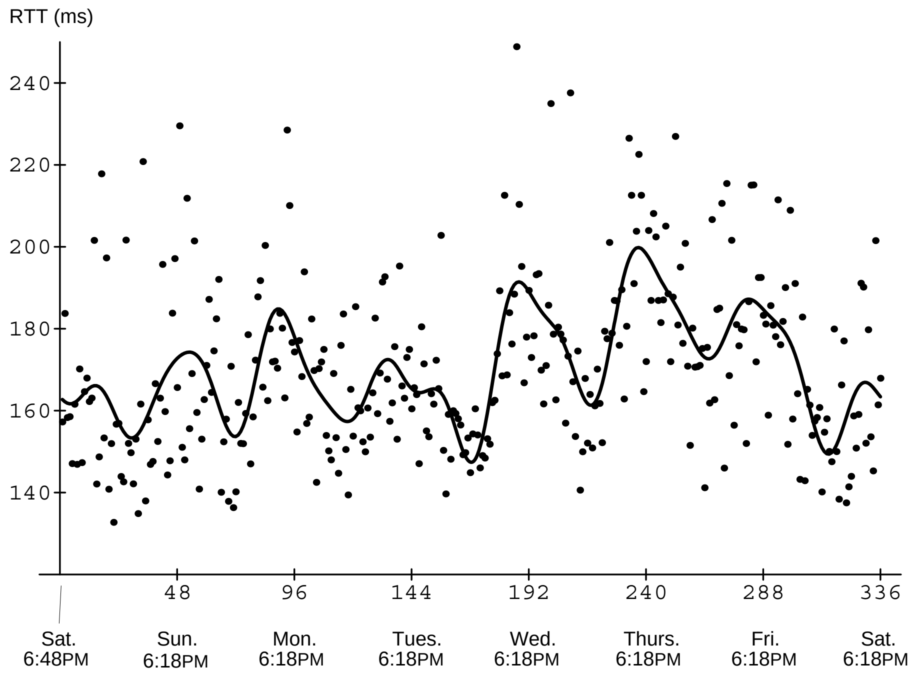
Using tools such as `ping` and RPC mirror we can measure the response-time of message passing over a WAIN. By sending messages repeatedly we obtain the effective bandwidth of the communication channel. We have already reported on the response-time of the IP layer in Table 1 of Section 3.2. Although we have measured the network effective bandwidth in the past we have not measured the effective bandwidth for either the Webster sample or the Camelot sample. In 1989, during the switchover from ARPANET to NSFNet, the effective bandwidth of Mach RPC facility between Columbia and CMU was about 11KB per second for 8KB messages (we were unable to send messages longer than 8KB).

4.3 Periodicity Analysis

Another potential source of variance is the different demands on the network at different times of the day. For example, Cheriton [5] has found networks busy just before lunch and in the mid-afternoon. The daily and weekly components correspond to our intuitive model of typical machine load, which oscillates according to people's working patterns. Much of our data show clear daily and weekly periods of busy activity, either in the network or the end machines or both.

Figure 6 shows the Camelot sample data accumulated on a weekly basis. The periodicity analysis starts with an accumulation of the points at the same time in the same day of the week (averaged into the same bucket). Then we use the FFT package of Mathematica to find the cycles in the data. The curve superimposed on the "raw" data in Figure 6 is drawn taking the 15 first FFT components. The curve reveals up to two cycles per day (Saturday and Tuesday), but most of the other days show only one peak at about 5pm Eastern Standard Time. Although the weekly period is better seen on a curve containing only 2 first FFT components, Figure 6 shows the contours of the smaller weekly component, with Wednesday, Thursday, and Friday having higher load than Saturday and Sunday. Probably the fast response time on Tuesdays is a statistical fluctuation.

4.4 Internet Link Availability



Camelot sample, ping layer, raw data, stacked mean of batch means.

Figure 6: Daily Periodicity in a Week

Experiment	**completed successfully**	**failed**	**total**	**failure rate**
`traceroute`	315	15	330	4.6%
`ping`	303	27	330	8.2%
mirror-RPC	292	38	330	12%
Webster	294	36	330	11%

Webster sample, all layers, raw data.

Table 6: Observed Failure Rates

One important statistic in any network experiment is the total number of failed experiments, which did not return data. Table 6 shows the failure rates of all four layers in the Webster sample. One can see that `traceroute` and `ping` had similar low failure rates, while mirror-RPC and Webster had similar high failure rates. This is not surprising, since `traceroute` and `ping` use the same low level IP protocol while mirror-RPC and Webster use the high-level RPC facility, thus sharing the same failure modes.

When `traceroute` fails, the Columbia—Washington path is completely blocked. In addition to the 4.6% of the time this happens, `ping` fails for another 3.6% of the time. Webster fails an additional 3% of the time, due to failures between IP and RPC layers, including one case of an invalid RPC port. We have not found the explanation for the two cases where mirror-RPC failed but Webster did not.

5 Application Analysis

5.1 Operating System Overhead

For mirror-RPC and RPC-based applications we have the problems discussed in Section 4.4 plus the timeout and retry nature handled automatically by the operating system's remote procedure call mechanism. While the number of retries due to lost or delayed packets carrying the RPC's is unknown, it can be estimated from the known number of packets for each RPC. In any case we would be interested in both the average behavior (as seen by the users) and the understanding of the underlying software layers. Data points that include the lengthy recovery are important for the user perception but not for systems analysis.

Although IP represents the smallest communications overhead in the Internet, most application programmers prefer higher level primitives. In Mach, the OS running on our measurement machines, the most popular communications facility is the Remote Procedure Call (RPC). There are more layers in the OS between RPC and IP, but we will simplify the experiment by concentrating on the lowest necessary layer (IP) and the highest OS primitive used by programmers (RPC). The idea is that IP results measure the network travel time, while the difference between RPC and IP captures the OS overhead.

There is a significant amount of work done by the OS kernel and run-time library for each RPC call. The additional cost (compared to IP) includes both CPU overhead and messages.

- IP is part of the kernel, but RPC is not. There is some kernel entrance/exit cost for RPC.
- RPC needs to marshal and unmarshal invocation arguments, possibly copying the message body.
- RPC requires more messages than `ping`. A Mach 2.5 RPC takes at least 2 packets (one each way) and more if a large message is divided into standard size packets. Each message has a TCP ack packet that is usually piggybacked to existing traffic. In addition, the first messages has an initial overhead of 2 packets.

Mach does not supply the equivalent of `ping` for the RPC layer. We wrote our own measurement tool, consisting of a client/server pair. The client communicates with the server through RPC and records the round-trip time, very much the same way as `ping`, but at the RPC level. This tool is called *mirror-RPC*, since the server simply reflects the

Data Type	Mean of Means	std.dev. of Means	Remain. Data
Raw	380 ms	666 ms	110%
Drop First	370 ms	617 ms	100%
Thres. 1 sec	233 ms	46 ms	98.0%
Albany/Cornell/Illinois	222 ms	37 ms	91.5%

Webster sample, RPC layer, mean of batches, filtered three ways.

Table 7: Refining mirror-RPC Data

Data Type	Mean of Means	std.dev. of Means	Remain. Data
mirror-RPC−`ping`	35.7 ms	46 ms	100%
Albany/Cornell/Illinois	35.5 ms	37 ms	72%

Data Type	Mean of Minima	std.dev. of Minima	Remain. Data
mirror-RPC−`ping`	17.3 ms	22 ms	100%
Albany/Cornell/Illinois	18.6 ms	10 ms	72%

Webster sample, RPC and ping layers, means of batches, filtered two ways.

Table 8: Mirror-RPC minus `ping`

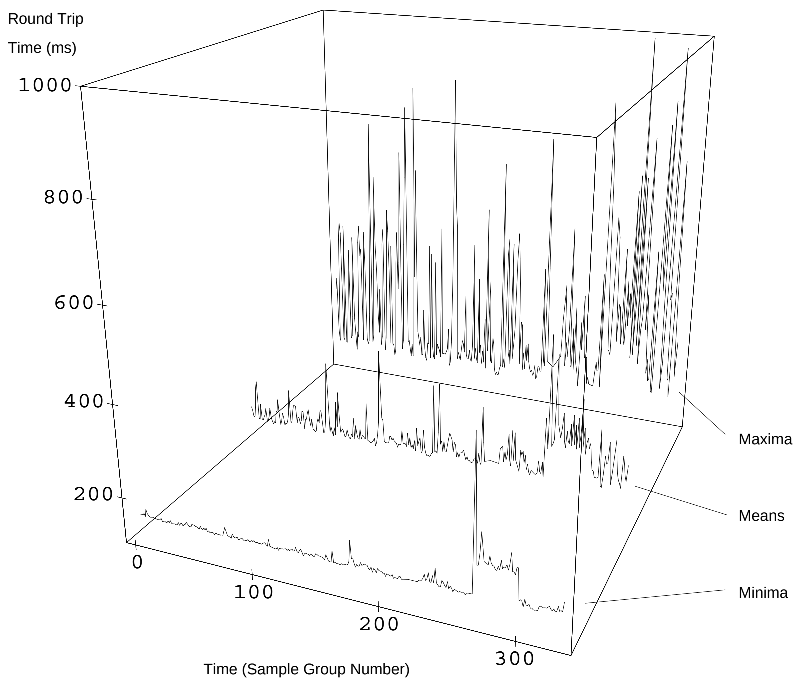
incoming message back to the client. Similar to the `ping` measurements, the mirror-RPC data collection happens every half an hour, coinciding with the `ping` experiments. For uniformity we take 11 samples (RPC round-trips) for each sampling period. The data treatment is the same.

To show the effectiveness of our data refinement method, we include a summary of data refinement from the two other layers. Table 7 summarizes the refinement of data for the middle layer, called mirror-RPC. We emphasize the clear similarity in improvements of Tables 1 and 7, even though they measure different layers of network software and several times difference in absolute numbers measured.

5.2 Layered Data Analysis

One of the important results of the LR methodology is the ability to derive performance information about each layer of the software being measured. The first step is to subtract the measured `ping` number from the mirror-RPC number, point by point. The idea is that since those number were obtained at the same time, they are correlated (see Section 3.3). As we can see from the top half of Table 8, subtracting the means of each experiment has a large standard deviation. The second half of Table 8 shows the subtraction done between the minima of each experiment instead of mean.

The main reason we prefer minima for inter-layer subtraction is the analogy with labo-



Webster sample, ping layer, raw data, filtered at 1000 ms threshold.

Figure 7: Comparison of Minima, Means, and Maxima

ratory experiments (such as [15]). In a laboratory, all the external interference is eliminated to yield consistently reproducible results. This process yields the minima as the final result. In our measurements, we cannot eliminate the external interference, but the minima approximates the situation since they show the least perturbation. Therefore, when analyzing the layers of system software we prefer the minima to means. To illustrate the significance of minima data, Figure 7 compares the minima, means, and maxima from the Webster sample.

The difference attributable to the RPC overhead thus calculated is about 18 milliseconds with a narrow confidence interval (row 3 and 4 of Table 8). This number comes quite close to laboratory measurements of Mach RPC. Duchamp [7] has reported a total Mach RPC round-trip time of 22 milliseconds between two IBM RT/125s connected through a token-ring LAN. Since the network, basic software releases, and machines are different, the measurements are not directly comparable. Section 3.5 explains the LAN control experiments that provide a more appropriate comparison.

Similar to subtracting `ping` numbers from mirror-RPC numbers, we can subtract the RPC cost from Webster cost. Table 9 shows it takes 40 milliseconds elapsed time for the

Data Type	Mean of Means	std.dev. of Means	Remain. Data
Webster−RPC	58 ms	62 ms	100%
Albany/Cornell/Illinois	56 ms	50 ms	72%

Data Type	Mean of Minima	std.dev. of Minima	Remain. Data
Webster−RPC	41 ms	18 ms	100%
Albany/Cornell/Illinois	40 ms	14 ms	72%

Webster sample, RPC and application layers, means of minima in each batch, filtered two ways.

Table 9: Webster Minus mirror-RPC

Network Type	Webster−RPC	RPC−`ping`	`ping`	Total
WAIN – breakdown	40 ms	19 ms	157 ms	216 ms
LAN – breakdown	48 ms	15 ms	4 ms	67 ms

Webster sample, all three layers, minima of batches.

Table 10: Adding Up Layers (all Minima)

Webster dictionary to look up a word. The actual layer-by-layer analysis is in Table 10.

One interesting observation from Table 10 is the difference in layer contributions between LAN and WAIN experiments. Self-interference is the most probably cause of this discrepancy, which is a topic of active research.

6 Related Work

6.1 Iterated Refinement Methodologies

Seo et al. [17] have measured the end-to-end performance of TCP/IP over the Atlantic Packet Satellite Network (SATNET). Like the LR methodology proposed here, the measurement methodology in the SATNET study is an interactive one. It is also concerned with end-to-end performance over the network. However, the SATNET study is limited to the TCP/IP layer, since they have a simpler environment.

Work by Mills [11] and Schwartz [16] have both studied applications across WAINs that are related in methodology to Layered Refinement. Although the focus of these works was not to measure the performance of applications, the methodology they employ is similar to LR in that both iterated the steps of the scientific method: build a model of the system, formulate hypotheses to capture sources of error, filter and interpret data, and refine the model accordingly.

The LR approach differs from the above works for we are interested in measuring the response time of layered software applications over a WAIN. For this purpose the LR methodology uses simultaneous multi-layer data collection to obtain information on each layer. Our methodology is more general than their work, since LR can be applied to any layered software application running over a WAIN.

A number of other studies (such as [2], [9] and [10]) look at patterns of traffic and reliability of nodes connected to WAINs. However, unlike LR, these studies do not attempt to measure the response time of applications running across WAINs since they are studying the network as a conduit for packets.

6.2 LAN Protocol Measurements

A study of the Firefly RPC [15] is similar in objective to LR since it measures the end-to-end performance over a LAN. However, unlike LR, this work measures only RPC in a controlled environment. The goal of LR is to simultaneously measure layers such as application, RPC and network to build an overall profile of end-to-end performance. In order to produce consistent and reproducible results, the Firefly RPC study sheds the load from the nodes in the experiments and also uses an isolated LAN. When experimenting over live WAINs, this is impossible. Consequently, a methodology such as LR is essential in order to make experimental results over WAINs consistent and reproducible.

Studies such as [19] and [5] measure the performance of transport services over LANs. However, these studies are concerned with the performance of the transport protocols in specific network environments. Layered Refinements is more concerned with end-to-end throughput as measured at the user-level. There has also been work done in studying the performance of TCP/IP in both LAN and WAIN environments [6, 4].

7 Conclusion

The problem of accurately measuring the performance of applications in a WAIN is important and difficult. First and foremost, it is impossible to control many important variables which directly affect performance, such as network load and packet path. This introduces large variances in measurements. Similarly, we have no control over the network configuration. This prevents us from using specialized hardware or software tools to measure the network. To aggravate the problem further, the underlying network topology, hardware, and software are constantly changing. Besides the hardware change examples we have mentioned in the paper, operating system and applications software are often updated. All of these problems make any systematic measurements of applications over WAINs difficult, as illustrated by the raw data in Figures 6 and 7.

We have developed the LR methodology during our three-year experience in measuring distributed WAIN applications. LR consists of two steps iterated several times. The first step is the simultaneous measurements of all layers of the application over an extended period of time. The second step is a careful analysis and filtering of data to identify the noise introduced into the measurements by “warm-up” and routing changes. The second step (data filtering) significantly improves the precision of data. For example, we have reduced the standard deviation of Webster sample by a factor of 10 through outlier elimination. The first step (simultaneous measurements) allows us to correlate the observations on different layers, so we can subtract the cost of lower layers from the higher layers. By combining the two steps we can obtain a better picture of the application at each layer such as IP, RPC, and applications.

Our methodology is simple and practical. It relies on user-level measurement tools and and does not use any special "hooks" or require privileged authorization in the underlying hardware or software. The measured packets are not required to carry additional information which would likely skew results. The result is an improvement in the quality of data which allows us to side-step large variances and thus analyze the true end-to-end performance of a variety of applications.

An interesting aspect of this work is its scientific flavor. Unlike the usual process of computer system building in a laboratory, we observe the behavior of a "natural" system, which we cannot control. To arrive at our results, we have to iterate through the traditional steps of establishing hypotheses, constructing a model, and accumulating enough data to confirm or refute the hypotheses. From this perspective, our methodology is very promising but far from complete. Currently we are working on more rigorous data analysis methods and design of better control experiments.

8 Acknowledgments

This project was supported by Digital Equipment Corporation, which provided two Microvaxes dedicated to the Columbia–CMU experiments through the External Research Program in 1987. The grant was supported and monitored by Philip Bernstein and Andrew Black, both from DEC Cambridge Research Lab. The equipment used in the Columbia–Washington experiments was provided by National Science Foundation through a CISE Instrumentation grant in 1989.

We gladly acknowledge the generous help from people at University of Washington and Carnegie–Mellon University, who made our measurements possible with equipment and support. At Washington our experiments are sponsored by Prof. John Zahorjan, with assistance from Eric Lundberg and Jan Sanislo. At CMU our experiments were sponsored by Prof. Alfred Spector, with assistance from Howard Wactlar, Mary Thompson, Peter Stout, George Michaels, and several other staff members and graduate students. Prof. Daniel Duchamp of Columbia University helped us keep the CMU end running after Prof. Spector's move to Transarc.

Many project students participated in monitoring of these measurements through the years. We note Mark Kennedy and Ramesh Balan for their industry and skill. Other project students include Jeff Alvidrez, Krishna Bala, Robert Cartolano, Akira Kawaguchi, and Manju Parekh.

References

[1] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young. Mach: A new kernel foundation for Unix development. In *Proceedings of the 1986 Usenix Conference*, pages 93–112. Usenix Association, 1986.

[2] Ramon Caceres. Measurements of wide area Internet traffic. Technical Report UCB/CSD 89/550, Computer Science Division (EECS), University of California, Berkeley, December 1989. PROGRES Report No. 89.12.

[3] J. Case, M. Fedor, M. Schoffstall, and C. Davin. A simple network managment protocol (SNMP). Technical Report RFC-1098, Network Working Group, DDN Network Information Center, SRI International, Menlo Park, CA, April 1989.

[4] S.T. Chanson, K. Ravindran, and S. Atkins. Performance evaluation of the ARPANET transmission control protocol in local area network environment. Technical Report 85-6, Department of Computer Science, University of British Columbia, 1984.

[5] D. Cheriton and C. Williamson. Network measurement of the VMTP request-response protocol in the V distributed system. In *Proceedings of the 1987 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 216–225. ACM/SIGMETRICS, May 1987.

[6] D.D. Clark, V. Jacobson, J. Romkey, and H. Salwen. An analysis of TCP processing overhead. *IEEE Transactions on Communications*, 27(6):23–29, June 1989.

[7] D.J. Duchamp. Analysis of transaction management performance. In *Proceedings of the Twelfth Symposium on Operating Systems Principles*, pages 177–190. ACM/SIGOPS, December 1989.

[8] R. Golding and Darrell D. E. Long. Accessing replicated data in a large-scale distributed system. Technical Report UCSC-CRL-91-01, Concurrent Systems Laboratory, Baskin Center for Computer Engineering & Information Sciences, University of California, Santa Cruz, January 1991.

[9] S.A. Heimlich. Traffic characterization of the nsfnet national backbone. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 257–258. ACM/SIGMETRICS, May 1990.

[10] D.D.E. Long, J.L. Carrol, and C.J. Park. A study of the reliability of Internet sites. Technical Report UCSC-CRL-90-46, Computer and Information Sciences, University of California, Santa Cruz, September 1990.

[11] D.L. Mills. Measured performance of the network time protocol in the Internet system. Technical Report Request for Comments 1128, Network Working Group, DDN Network Information Center, SRI International, Menlo Park, CA, October 1989. Mills is with University of Delaware.

[12] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1988.

[13] C. Pu and F. Korz. Measurement of Camelot performance over the Internet. Technical Report CUCS-001-91, Department of Computer Science, Columbia University, February 1991.

[14] Marty Schoffstall. E-mail posted on columbia university bboard. Explained the NYSERNet/NSFNet connectivity disruption., November 1990.

[15] M. Schroeder and M. Burrows. Performance of Firefly RPC. *ACM Transactions on Computer Systems*, 8(1):1–17, February 1990.

[16] M. F. Schwartz and D. C. M. Wood. A measurement study of organizational properties in the global electronic mail community. Technical Report CU-CS-482-90, Department of Computer Science, University of Colorado, August 1990.

[17] K. Seo, J. Crowcroft, P. Spilling, J. Laws, and J. Leddy. Distributed testing and measurement across the Atlantic packet satellite network (SATNET). In *Proceedings of the 1988 ACM/SIGCOMM Symposium*, pages 235–246. ACM/SIGCOMM, August 1988.

[18] A.Z. Spector, D.S. Thompson, R.F. Pausch, Eppinger J.L., D. Duchamp, R.P. Draves, D.S. Daniels, and J.J. Bloch. Camelot: A distributed transaction facility for Mach and the Internet - an interim report. Technical Report CMU-CS-87-129, Computer Science Department, Carnegie-Mellon University, June 1987.

[19] L. Svobodova. Measured performance of transport services in LANs. *Computer Networks and ISDN Systems*, 18(1):31–45, November 1989.

[20] J.W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, 1977.