An Active Measurement System for Shared Environments

Joel Sommers
Colgate University
jsommers@colgate.edu

Paul Barford
University of Wisconsin-Madison
pb@cs.wisc.edu

ABSTRACT

Testbeds composed of end hosts deployed across the Internet enable researchers to simultaneously conduct a wide variety of experiments. Active measurement studies of Internet path properties that require precisely crafted probe streams can be problematic in these environments. The reason is that load on the host systems from concurrently executing experiments (as is typical in PlanetLab) can significantly alter probe stream timings. In this paper we measure and characterize how packet streams from our local PlanetLab nodes are affected by experimental concurrency. We find that the effects can be extreme. We then set up a simple PlanetLab deployment in a laboratory testbed to evaluate these effects in a controlled fashion. We find that even relatively low load levels can cause serious problems in probe streams. Based on these results, we develop a novel system called MAD that can operate as a Linux kernel module or as a stand-alone daemon to support real-time scheduling of probe streams. MAD coordinates probe packet emission for all active measurement experiments on a node. We demonstrate the capabilities of MAD, showing that it performs effectively even under very high levels of multiplexing and host system load.

Categories and Subject Descriptors: C.2.3 [Network Operations]: Network management, Network monitoring, C.2.5 [Local and Wide-Area Networks]: Internet (e.g., TCP/IP), C.4 [Performance of Systems]: Measurement Techniques

General Terms: Design, Experimentation, Measurement, Performance

Keywords: Active Measurement, MAD

1. INTRODUCTION

Several key challenges for networking research were specified in the 2001 National Research Council report entitled "Looking Over the Fence At Networks: A Neighbor's View of Networking Research" [18]. Among these was to develop an understanding of Internet structure and behavior through empirical measurement, including the grand challenge of capturing "a day in the life of the Internet". The lack of an intrinsic and openly available measurement capability in the Internet implies that a widely deployed infrastructure capable of different types of measurement would be required as a critical component for addressing these challenges.

The need for Internet testbeds capable of supporting accurate active measurements has been apparent for quite some time, and has resulted in deployment and operation of several different infrastructures over the years. A well-known early example was the National Internet Measurement Infrastructure (NIMI), which was composed of guest accounts on 35 end hosts located primarily in the US and Europe [29]. The NIMI effort helped to crystallize the challenges associated with using and operating Internet measurement testbeds. Among these challenges is the need for a large number of diverse sites (*e.g.*, commodity versus research network-attached, geography, last-hop bandwidth) such that a wide range of conditions is likely to be experienced across the paths between measurement nodes.

Perhaps the most prominent Internet testbed today, PlanetLab, is comprised of 780 end hosts deployed at 382 different sites all over the world [14]. PlanetLab is a canonical example of an openly available network testbed that is designed to support many different types of network-related experiments at the same time. The fundamental design requirement of simultaneous experimental support has a very important implication. Resource scheduling in these environments, at the individual host level or globally across the testbed, poses a particularly difficult problem. Thus, any experiment that has even modest timing or coordination requirements can be difficult or impossible to run in shared testbed environments. An important class of experiments that have until now been largely excluded from shared network testbeds are those that use active probe tools to measure end-to-end path properties such as delay, loss, capacity and available bandwidth.

In this paper we address the problem of how to extend shared network testbeds to enable accurate active measurement studies that use tools with fine-grained timing requirements. We begin by assessing the magnitude of the bias introduced by shared network testbeds. We instrument the two PlanetLab nodes at our own site with time-synchronized, hardware-based packet capture systems and conduct a series of active measurement experiments between those two nodes (*i.e.,* such that there are no unknown network effects). We find that even in this simple local area environment, very large bias in active probe streams is common. For example, we found that there are often periods of multiple seconds in which packet loss is measured on end hosts while there is zero packet loss in the network. Likewise, RTT delays of over 50 milliseconds are not uncommon when the true delay between the two hosts as measured by the hardware-based systems is on the order of about 100 microseconds.

To address this problem, we develop the multi-user active measurement system (MAD) that is designed to support real-time schedul-

IMC'07, October 24–26, 2007, San Diego, California, USA.

ing of active probe streams. The key requirements for MAD include accurate transmission/receipt of arbitrary probe streams, support for simultaneous experiments, low impact on the host system, ease of use and security. We designed MAD as a service that can be accessed by applications through a simple active measurement specification language. Authorized users specify their probe process to MAD, which then coordinates transmission of probe streams from multiple experiments using the real-time features of newer Linux kernels to gain access to resources at the highest priority level. (Note that there is also a MAD receiver module and a MAD reflector module that are used for one-way and round trip measurement tools, respectively.) An additional requirement in order to facilitate deployment and use of MAD is that it not necessitate modifications to the host OS. As such we implemented MAD so that it could be run either as a stand-alone daemon or as a Linux kernel module.

We evaluate the capabilities of MAD in a laboratory testbed that includes a local area PlanetLab deployment. This environment enables us to control and measure the contention effects due to multiplexing experiments both with and without MAD. We begin by establishing a quantitative baseline of the extent to which load generated by simultaneous experiments can bias active probe streams and lead to inaccurate inference of path properties. We then show that with MAD, highly accurate measurements can be made even under extremely heavy system loads (regardless of whether MAD runs as an in-kernel process). Next, we conduct similar experiments on raw, unvirtualized operating systems without PlanetLab and find that although the effects are less severe, MAD can still improve measurement accuracy in a meaningful way. Finally, we demonstrate the scalability of MAD through a series of microbenchmark experiments and show that it is able to support a relatively large number of simultaneous measurement experiments with extremely low impact on system resources.

In summary, the contributions of this paper are *(i)* characterizing the extent to which active probe-based measurement can be skewed in PlanetLab, *(ii)* the design and implementation of MAD—a real-time scheduling system for accurate active probe-based measurement. We believe that MAD effectively addresses an important deficiency in current end host-based Internet testbeds that largely precludes, or at least casts doubt on, their use for empirical studies of path properties using active probe tools. By virtue of its implementation, it is our hope that MAD can be widely deployed and used. We also believe that the design of MAD can inform future testbed development (*e.g.,* GENI [7]) as well as infrastructures used to measure path properties in operational environments (*e.g.,* for SLA compliance monitoring [40]).

The remainder of this paper is organized as follows. In Section 2, we discuss studies related to our own. In Section 3, we present findings from our analysis of measurements taken from our local PlanetLab nodes. We describe the details of MAD's design, implementation and use in Section 4. The results of our laboratory evaluation of MAD are presented in Section 5, and in Section 6 we summarize and discuss future directions for our work.

2. RELATED WORK

Host-based testbeds deployed in the Internet enable implementations of network applications, protocols, and measurement methodologies to be evaluated over live end-to-end paths. These testbeds can also be useful for gauging and characterizing Internet structure and end-to-end performance from multiple vantage points. Some testbeds in the past have been developed for use by a single organization for a specific set of objectives, *e.g.*, Surveyor [22], while others have been designed from the outset to be general purpose and shared with a wider set of researchers *e.g.*, RON [10] and PlanetLab [14, 30]. There have been measurement studies conducted on shared testbeds including RON and PlanetLab, *e.g.*, [12, 31, 44], and specialized systems have been developed for these testbeds to perform or assist with various types of measurements such as ScriptRoute [43].

PlanetLab in particular has been deployed with the goal of serving a large community of researchers, and has also spawned other regional instantiations [1, 4]. PlanetLab uses virtualization techniques to isolate users from one another [14, 36]. It virtualizes a host system at the system call level, similar to the way BSD Jails work [35], but a number of other approaches are possible, *e.g.*, [13, 33]. It is important to note that in heavily-used Internet testbeds like PlanetLab there is a direct correlation between contention for resources and the level of experimental multiplexing that is allowed.

One study with similarities to ours presents guidelines in the form of myths and realities for performing measurement studies on PlanetLab [42]. One guideline of relevance is to use system interfaces for obtaining kernel timestamps for probe packets. Our work also uses these techniques. Another suggestion is to use the ScriptRoute [43] system for defining and executing the probe process. Our work bears similarity to the ScriptRoute system in that we also design a measurement service, one component of which is a programmatic interface for specifying the probe process. However, our work takes a fundamentally different approach by focusing not only on providing a flexible measurement service, but also on measurement accuracy. We assume that probe processes operate according to a discrete time clock, enabling optimizations to be made over all running measurement processes. ScriptRoute makes no such restrictions. Thus, the scheduling requirements of ScriptRoute are significantly different from our system.

Our system design takes advantage of the real-time scheduling capabilities that have been incorporated into the main Linux kernel source [5, 6]. These capabilities are similar to some of the features used in the related study by Pásztor and Veitch in which a real-time version of Linux along with techniques they devise for improving scheduling and timestamp fidelity are employed [27, 28]. Our work relates to other efforts in the network research community to improve the accuracy and precision of timing-sensitive applications using commodity systems [2, 11, 45].

Active measurement of end-to-end delay and loss characteristics have a long history within the network measurement community. The most well-known early study of these characteristics was reported by Bolot in [16]. While methodologies for measuring these quantities have been standardized by the IETF [8, 9], improvements to these techniques continue to be developed [38, 40]. Of particular relevance is the work by Sommers *et al.* that highlights the need for real-time probe management in a multi-objective measurement context [39]. Our work addresses a different problem: the impact of resource sharing on active probe streams, and develops a generalized mechanism to address the problem.

Using pairs of closely-spaced packets to estimate network quantities also has a long and rich history. Jacobson's work on TCP congestion control was an early exposition of the possibility of how a pair of closely-spaced packets can reveal bottleneck link capacity [20]. Other researchers have explored specific algorithms using this technique to perform capacity estimation, *e.g.*, [23, 24, 25], and for estimating end-to-end available bandwidth, *e.g.*, [32, 44]. Difficulties in creating accurate packet streams that match the probe models has been highlighted in many of these studies. For example, Carter and Crovella grappled with issues related to probe timing fidelity in developing active probe tools to measure bottleneck link speeds [17].

3. THE CURRENT SITUATION

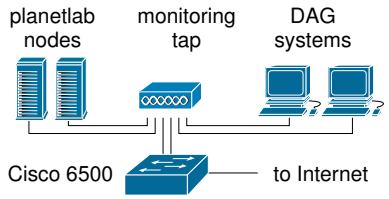
We first investigate the accuracy characteristics of three types of active measurements with stringent timing requirements using the widely-used PlanetLab shared infrastructure.

3.1 Experiments

Normally, a user of PlanetLab does *not* have access to so-called ground truth measures between an arbitrary pair of PlanetLab nodes. Since the goal of our experiments is to evaluate the accuracy of certain types of active measurements, this issue is of critical importance. Other researchers have used indirect means for addressing this problem (*e.g.*, by introducing known traffic to disturb the system as in [44]), or by using available coarse-grained SNMP data. These methods generally cannot give the kind of accuracy and reliability required for our study.

Our approach was to use the two PlanetLab nodes over which we have complete control of the networking infrastructure: `planetlab1` and `planetlab2` in the domain `cs.wisc.edu`. For ground truth measures, we introduced passive network taps and a pair of synchronized Endace 4.3GE DAG cards for capturing both inbound and outbound probe traffic for each of the PlanetLab nodes, as depicted in Figure 1.

Our two PlanetLab nodes are identical Dell Precision 340's with 1.8 GHz Pentium 4 processors and 2 GB RAM. They were running version 4.0 of the PlanetLab software with Linux kernels derived from version 2.6.12. We installed Intel/Pro 1000 Gigabit Ethernet NICs on each host to accommodate our passive monitoring system. Interrupt coalescence was disabled for our experiments. No other changes were made to these two hosts.



1: Setup for experiments using live PlanetLab nodes.

We used three active measurement algorithms on these two hosts to examine how system load and multiplexing can affect measurement accuracy: round-trip delay probes, packet-pair probes, and BADABING loss probes. Each of these algorithms was implemented to use the User Datagram Protocol and to run according to a discrete time clock. We used the `SO_TIMESTAMP` option of `setsockopt` to obtain kernel timestamps on receipt of packets. These three measurement algorithms were chosen to be representatives of standard active measurements relying on a high degree of accuracy in both probe stream transmission and reception. A description of each algorithm along with its relevant parameters and accuracy requirements is shown in Table 1. The discrete time interval used in the results that we report below was 5 milliseconds.

We collected data from 11 April 2007 to 2 May 2007. We initiated experiments every 4 to 6 hours, running each measurement algorithm for 10 minutes each. We simultaneously collected CPU utilization information using `vmstat` and packet header traces using the DAG systems. Since we ran each algorithm according to a discrete time clock, we also recorded the number of times our measurement process was unable to send probes at an intended time slot (*i.e.*, "slipped" a time slot).

While we collected measurements, there were typically 40–50 active slices on each host, resulting in hundreds of active network connections and very little idle CPU time. Quite often, CPU utilization was at 100% for each node and there were more than 1000 network connections. A survey of other PlanetLab nodes using CoMon [26] shows that the heavy load we observed is not abnormal. Somewhat surprisingly, there was relatively little network traffic produced by each of these machines. On average, there was about 400 Kb/s inbound traffic to each host, and about 500 Kb/s outbound from each host. (These numbers varied between less than 100 Kb/s to a little more than 1 Mb/s over periods we collected measurements.)

3.2 Results

Figure 2 shows timeseries plots of the median, 90th, and 95th percentile round-trip delays for two example measurement periods. Table 2 shows median, 90th, 95th, and 99th percentile delays for a larger subset of measurements. The results shown in the figures and table are qualitatively representative of all results we measured.

In Figure 2 and in Table 2 we first see alarmingly high delay values for *nearly all* quantiles displayed. For example, we see a median RTT of 9 milliseconds in the 11 April 12:20 measurements. This value is surprising since these two machines are colocated, with one switch between them. Second, we observe excessive delay values in the upper quantiles, sometimes larger than 100 milliseconds. (In some measurement periods we measured *maximum* delays on the order of 5 seconds.) Finally, we see a high degree of variability in the measurements, in some cases even in the median delay (*e.g.*, in Figure 2a).

It is important to state that in all cases we measured the network delay of the probes using the DAG cards (*i.e.*, not including delay introduced at the end hosts) to be on the order of 100 microseconds. Clearly, these RTT measurements are very poor indications of the true delay between these two PlanetLab hosts.

We now examine packet loss characteristics between the two live PlanetLab systems. Note that in the experiments using BADABING, we disabled the one-way delay congestion inference mechanisms in the tool, and thus use only *actual* indications of packet loss. Since BADABING sends three packets back-to-back as a probe, only one packet must be lost in order for a probe to be marked as having experienced loss.

Figure 3 shows timeseries of periods during which consecutive probes experience loss. Plots are shown from two representative measurement periods. Notice from the plots that there is a wide range of durations over which consecutive losses are measured, from very short periods (*e.g.*, in Figure 3a) to very long periods (*e.g.*, in Figure 3b). Note also that the rate of all our measurements, including loss, was less than the bandwidth limitation imposed on operational PlanetLab nodes.

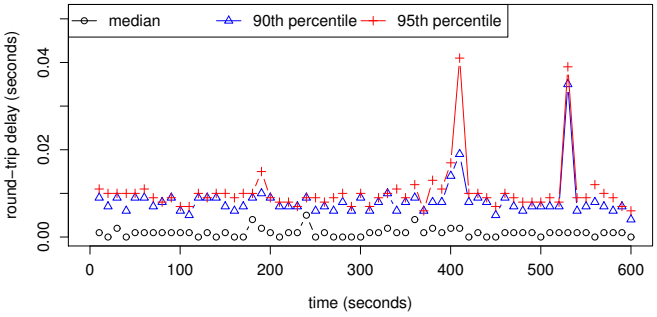
Table 3 shows representative results from the BADABING experiments for additional measurement periods. The table shows the frequency and duration of loss episodes reported by BADABING, as well as a loss rate estimate based on the heuristic of [39]. We see that there are estimated loss durations on the order of multiple seconds, as suggested by Figure 3. We also see that the loss rate estimates are similar to the frequency estimates. The reason for this effect is that the loss rate of all probe packets during loss episodes is close to 1. That is, during loss episodes, *nearly all probe packets are lost*. As suggested by results in the table, there were very few measurement periods overall in which we measured zero loss.

As with our round-trip delay experiments, we measured the true packet loss between the PlanetLab hosts using the DAG systems and found in all cases that there was no packet loss (*i.e.*, *all* probes are observed on the wire just after transmission and just prior to reception by a PlanetLab host). Thus, all loss indications that we

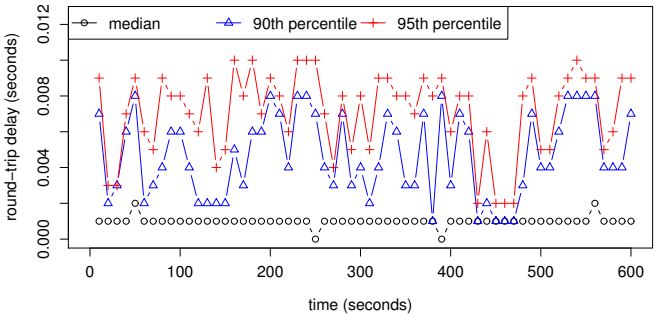
1: Description of the probe algorithms and their accuracy requirements used in experiments.

Probe algorithm and description	Accuracy Requirements
Round-trip delay measurements consisted of 100 byte UDP probes, sent periodically at 100 millisecond intervals. A process on the remote host bounced probes back to the sender, which then recorded round-trip delay.	For RTT measurements, timestamps should be applied in such a way as to most accurately reflect network delays, rather than operating-system introduced delay. Probes should be sent according to the intended process (*i.e.*, periodic probes, or probes sent according to a geometric process), as closely as possible in order to limit measurement bias. Since timestamps are applied by a single host, there are no time synchronization requirements.
Packet-pair measurements consisted of 1500 byte UDP probes sent with an intended spacing between packets of 120 microseconds (*i.e.*, back-to-back, assuming a capacity of 100 Mb/s). A packet pair was sent at a given time slot with independent probability $p = 0.2$, resulting in a geometric process of packet pair transmissions.	Packet pair spacings should follow the intended spacing as closely as possible on transmission and measured spacings at a receiver should accurately reflect the spacing of the packet pair. Inferences are typically made based on small perturbations in spacings, *e.g.*, on the order of tens or hundreds of microseconds, so measurement accuracy is critical to overall algorithmic accuracy (*e.g.*, see [41]). Timestamps applied to packets should accurately reflect the actual spacing of packets upon sending, and upon receipt. The intended probe process should be accurately followed in order to limit measurement bias.
BADABING loss measurements. BADABING sends pairs of probes at time slots i and $i+1$ initiated with independent probability p which we set to 0.3. Each probe consisted of three packets, each of 600 bytes, sent back-to-back (as quickly as the host system would allow) as described in Sommers *et al.* [38].	A high degree of accuracy in the probe emission process is required for BADABING. Probe pairs are sent at consecutive discrete time slots, with the discrete time interval set (by default) at 5 milliseconds. BADABING also uses measurements of one-way delay to infer congestion along a path. Note that in this paper, we turn off this inference capability of BADABING, relying only on *actual* packet loss.

measured with BADABING probes were confined to the hosts themselves and were completely spurious from a network measurement perspective. These results clearly pose a significant problem not only to network inference algorithms, but to any experiments deployed on shared testbeds that may be sensitive to loss (*e.g.*, throughput experiments of different applications or transport protocols).



(a) 23 April, 12:20–12:29



(b) 27 April, 08:20–08:29

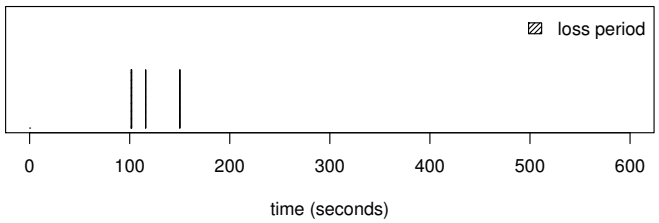
2: Timeseries of round-trip delays for two example measurement periods. Median, 90th, and 95th percentile delays are shown for each 10 minute measurement period.

For the packet pair measurements, we focus on two types of errors: errors in the initial spacing of a packet pair and errors in timestamps. With respect to errors in initial spacings, we compare the difference in spacing that a process attempts to achieve, and the actual spacing obtained. Typically, initial packet pair spacings are on the order of tens or hundreds of microseconds (*e.g.*, the spacing of a back-to-back pair of 1500 byte packets on Fast Ethernet is about 120 microseconds). In software, these fine timescale spacings are generally produced by busy-waiting between system calls to emit packets since most commodity operating system timer functions cannot deliver accuracy on these timescales. For timestamping error, we examine the differences between the timestamps applied by a sending user process to a packet pair and the actual spacing of the packet pair on the wire as measured using the DAG cards. We also examine the differences between the timestamps applied to packet pairs by a receiver process, and the actual spacing of the packet pair just prior to reception by the receiving host, also measured using the DAG cards.

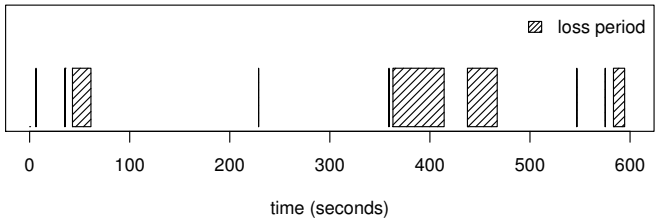
We found that errors on initial spacings are consistent with the results from [41], namely, that while there can be variability introduced in spacings due to system load and the operating system scheduler, the mean error is close to zero when considering a few

2: Quantiles of the delay distribution measured on live PlanetLab nodes. Results shown from a qualitatively representative selection of experiments.

Date/Time	Delay Quantiles (seconds)			
	50	90	95	99
11 April 12:20	0.009	0.057	0.071	0.100
12 April 06:20	0.000	0.002	0.006	0.022
14 April 06:20	0.000	0.002	0.006	0.018
14 April 18:20	0.000	0.002	0.006	0.014
16 April 18:20	0.002	0.009	0.010	0.016
18 April 06:20	0.000	0.008	0.010	0.016
18 April 18:20	0.000	0.007	0.009	0.013
23 April 06:20	0.001	0.008	0.010	0.017
23 April 12:20	0.001	0.008	0.010	0.020
26 April 08:20	0.000	0.009	0.029	0.100
26 April 16:20	0.000	0.008	0.018	0.065
26 April 20:20	0.001	0.005	0.008	0.012
27 April 08:20	0.001	0.005	0.008	0.012
30 April 08:20	0.000	0.008	0.042	0.094
30 April 12:20	0.000	0.006	0.010	0.046



(a) 12 April, 00:30–00:39

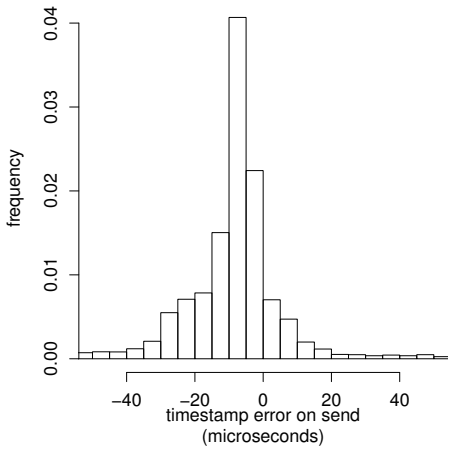


(b) 30 April, 06:30–06:39

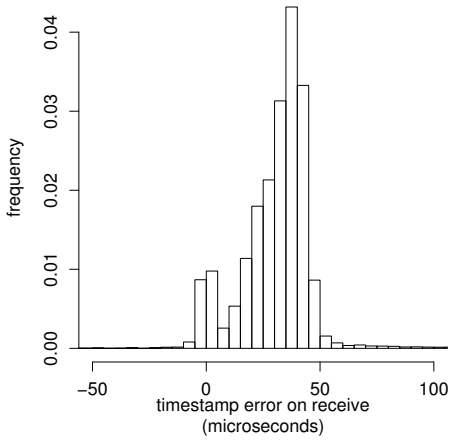
3: Timeseries of loss periods for two example measurement periods. Shaded blocks represent time periods during which all probes suffered at least one packet loss (*i.e.*, of the three packets in a BADABING probe, at most two were successfully transmitted).

3: BADABING results for live PlanetLab nodes. Results shown from a qualitatively representative selection of experiments.

Date/Time	Badabing Results		
	Loss Frequency	Loss Duration (seconds)	Loss Rate Estimate
14 April 18:30	0.0000	0.000	0.0000
15 April 06:30	0.0000	0.000	0.0000
15 April 18:30	0.0014	0.118	0.0014
16 April 06:30	0.1334	3.671	0.1333
16 April 18:30	0.1219	0.911	0.1217
17 April 08:30	0.0000	0.000	0.0000
17 April 18:30	0.0657	1.735	0.0656
18 April 06:30	0.0687	1.449	0.0687
18 April 18:30	0.0439	1.165	0.0438
19 April 06:30	0.0553	5.825	0.0554
19 April 18:30	0.0000	0.000	0.0000
20 April 06:30	0.0320	1.685	0.0320
20 April 18:30	0.0160	1.695	0.0159
21 April 06:30	0.0213	1.120	0.0212
21 April 18:30	0.0675	1.780	0.0674



(a) Histogram of timestamp errors for sending packet pairs.



(b) Histogram of timestamp errors for receiving packet pairs.

4: Histograms of send and receive timestamp errors for packet pair experiments on live PlanetLab hosts. Data collected on 1 May 2007 at 11:40–11:49. Results from other measurement periods are qualitatively similar.

tens of packet pairs. Similarly, we found that the timestamping error upon sending a packet pair has close to zero mean when considering a few tens of packet pairs. Figure 4a shows a representative histogram from 1 May 2007 of timestamp errors on sending a packet pair. These results are consistent with the notion that when a process gets scheduled, it generally retains the processor for the relatively short duration that it needs to busy-wait between sending packets.

For timestamp errors on receiving packet pairs we found that distribution of errors often does not have zero mean. Figure 4b shows a histogram of receive timestamp errors from data collected on 1 May 2007. The distribution is clearly shifted away from zero, with a peak around 40 microseconds. Note that for these measurements, we used system call interfaces in Linux to obtain probe packet timestamps from the kernel, much as Spruce does [44]. Clearly, there is a consistent expansion of packet pair spacings as they arrive at the destination host and pass through the hardware and operating system to the user process.

As a calibration check on our packet pair measurements, we used the Spruce available bandwidth measurement tool which sends 100 pairs of packets to derive an estimate of available bandwidth [44], also collecting DAG packet traces while we ran Spruce. Since the detailed timings of packet pairs can be affected by specific programming constructs, we wanted to be sure that our results were not specific to the way in which our code was written. (In fact, our code was developed from the BADABING code base.) We found that the results obtained from using Spruce were consistent with those described above.

Finally, we show results for discrete clock scheduling errors for each probe algorithm. Table 4 shows the number of missed time slots *i.e.*, probe sending process woke up after it should have, for measurement periods on the 26th of April. We can see from these results that many times there was significant deviation from the intended probe schedule. (Note that since the discrete time interval is 5 milliseconds, 1000 slips implies a 5 second cumulative difference in the intended probe process versus what is realized.)

For each of the three measurement algorithms, we also experimented using the Sirius calendar scheduling system available through the PlanetLab web interface, which should result in higher priority given to the slice in which the measurement tools were run and potentially better accuracy. We did not observe any qualitative difference in the measurement results.

4: Discrete clock scheduling errors (time slot misses) for measurement periods on 26 April for three measurement algorithms. Total number of time slots in experiment is 120,000. Results are representative of other measurement periods.

Date / Time	Round-trip Delay	Packet Pair	BADABING
26 April 00:20	4633	5797	5411
26 April 04:20	547	6181	1208
26 April 08:20	4747	6989	2361
26 April 12:20	34	8667	8645
26 April 16:20	3866	20638	6287
26 April 20:20	6540	7991	5408

4. A MULTI-USER ACTIVE MEASUREMENT SYSTEM

In this section we first discuss general requirements for a system designed to enable accurate active measurement for users of a shared testbed environment. We then describe key aspects of the design of MAD, a multi-user active measurement daemon. Lastly, we discuss specific issues related to the implementation of MAD.

4.1 Requirements

Results from our experiments on the live PlanetLab nodes described in § 3 suggest that one way to improve the accuracy of active probe streams would be to implement aspects of required measurement functionality at lower levels in a host system. This idea is a key aspect of the classic end-to-end arguments [34], and motivates our requirements and design in this paper.

Our list of requirements for a system to facilitate accurate active network measurements in a shared testbed is as follows:

1. **Accuracy.** First and foremost, network measurements in shared environments should be accurate. They should reliably reflect the state of the network at the time of probing, rather than effects due to host system load. Our results using three standard active measurement techniques on two live PlanetLab hosts in § 3 show that inaccuracies can be significant for these probe methodologies when running in a shared environment that is virtualized at the system call interface level. As part of requiring accurate measurements, timestamps should be as accurate as possible, at least matching accuracy of a lightly-loaded non-shared host. Not all widely distributed shared testbeds use virtualization techniques, *e.g.*, RON [10], so it is important that measurements be accurate in both virtualized and non-virtualized environments.

2. **Permit Multiple, Simultaneous Users.** The measurement system should permit simultaneous use by multiple users. One way to meet this requirement is to leverage the notion of multi-objective probing described in [39,40]. Specifically, we first assume that all probe algorithms operate according to a discrete time slot. If multiple, simultaneously operating algorithms attempt to send a probe at the same time slot, a single probe can be marked according to the algorithms to which it applies. Sending a single, tagged probe can also reduce measurement bandwidth requirements, as discussed in [40]. Rather than considering multiple, simultaneous *algorithms*, we can consider multiple *users*. Indeed, multiple incarnations of the same algorithm, *e.g.*, periodic probes at a given frequency, should be able to operate at the same time on behalf of multiple users of such a system.

3. **Low Impact on Host System.** The system for coordinating measurements on a host in a shared testbed should require only enough processor, memory, and network resources to support accurate measurements. In addition, the system should scale to facilitate active measurement for a reasonably large number of simultaneous users.

4. **Permit Flexibility in Specifying the Probe Process.** The network measurement community continues to produce novel active measurement algorithms each year. Embedding a specific probe algorithm, or even a set of algorithms, in a shared measurement system is undesirable. Apart from the restriction of requiring an algorithm to operate in discrete time, the measurement system should permit a great deal of flexibility in being able to specify the probe process, from simple periodic probes, to more elaborate probe algorithms such as BADABING.

5. **Provide Secure Access to Users.** A system for supporting active network measurements in a shared environment should not permit itself to be the launch pad for denial-of-service attacks. The system should provide, at minimum, authentication and authorization functions to identify users and provide appropriate access to the measurement facility.

6. **Provide Limits on Probe Traffic.** In addition to authentication and authorization, accounting is an important facility that the system should provide. In particular, the system should permit probe traffic quotas to be specified. For example, it may be desirable to impose an upper bound on traffic introduced by a user over a given time interval between two hosts or across the entire system.

7. **Provide Interfaces for Reporting Self-Measurement.** The system should allow users and administrators a window into its current operating state. For example, it may be possible to expose information to users about possible inaccuracies in measurements, such as scheduling errors in the case of discrete-time measurements, or other problems that can be ascertained by the measurement system itself.

8. **No Modifications to the OS Kernel.** Finally, if such a system is to be widely deployed, it should not require any changes to the host operating system. While benefits in accuracy may be obtained by avoiding software layers, or by reimplementing some existing functionality in a highly simplified manner, such changes might represent a significant barrier to wide adoption.

4.2 Design

We now describe the design of a system to support multi-user active measurements, MAD. We focus on the key problem of providing a flexible probe process specification for multiple, simultaneous users.

The basic approach we take is for users to specify the probe algorithm using a restricted language. Code written in this restricted language is executed as a callback. Constructs in the language exist to schedule future callbacks, to schedule probes to be sent at a specific discrete time slot, and to perform basic arithmetic (integer)

operations. Our approach bears similarities to a virtual machine execution environment and to the approaches of the Exokernel and SPIN operating systems that allow application-specific code fragments to be downloaded to a kernel execution environment [15,19]. In the latter context, the execution kernel is MAD itself, and the application-specific code implements the key aspects of a probe algorithm, *e.g.*, a periodic probe, or geometrically distributed probe pairs.

To specify a probe algorithm, MAD provides an assembly-like language, MADcode. It currently contains five operations, which can be easily translated to bytecode that is executed by MAD. Four virtual registers, r0–r3, are available for use: two have pre-defined uses and two are available for algorithm state maintenance or temporary storage, if necessary.

Upon download of MADcode by a user, the code is executed at the next possible time slot. Upon execution of user bytecode, register r0 contains the current time slot number and register r1 contains the current probe sequence number. Registers r2 and r3 are available for arbitrary use. After executing user bytecode at callback time, registers r1–r3 are saved for the next time the bytecode is executed. Any change to r0 is discarded. Thus, user code can control its own sequence through register r1 (with a maximum sequence value of 2^{16}). Note that the sequence refers to a probe, which may itself consist of multiple packets (*e.g.*, in the case of packet pairs and BADABING). Individual packets of a probe have an additional sequence number (with smaller range), currently assigned by MAD. Other probe process parameters, such as probe packet size, the number of packets emitted per probe, the spacing of those packets, and addressing information are currently specified when a user initiates a request to start a measurement experiment.

Table 5 shows the five operations that are currently defined in MAD. All MADcode operations are on integers—no floating point operations are allowed. In addition, there are no branch or looping instructions, eliminating the possibility for infinite loops or other common programming errors. As we gain experience with implementing different active measurement algorithms using MAD, we may find it necessary to expand the set of operations in MADcode.

5: MADcode instruction set.

Operation	Description
add rx, arg	Add the contents of rx and arg, placing the result in rx. arg may be a literal (denoted by a pound symbol before the literal integer) or may be another register.
load rx, arg	Copy the value from arg into register rx. arg may be a literal or a register.
geom rx, arg	Store a geometrically distributed random deviate with parameter $p = arg/100$ in rx. arg may be a literal or a register.
schedule_probe rx, ry	Schedule a probe to be sent at time slot rx with sequence number ry.
schedule_callback rx	Schedule a callback for time slot rx.

MADcode to implement the three measurement algorithms shown in Table 1 is shown in Listings 1, 2, and 3. The code for the periodic probe proceeds by simply adding a fixed number of time slots to the current slot number (lines 5 and 8), incrementing the probe sequence number (line 10), then scheduling a probe and the next callback. The geometrically distributed probe algorithm in Listing 2 is very similar to the periodic probe except that the number of time slots between probes is obtained through a call to the **geom** operation (line 5). This operation returns an integer representing the number of time slots until the next probe emission (and callback) drawn from a geometric distribution. The **geom** operation assumes a denominator of 100 for its parameter p. The numerator is specified as the second argument to the operation. The code for implementing the BADABING algorithm is shown in Listing 3. BADABING emits pairs of probes at time slots i and $i+1$, initiated with independent probability p at time slot i. The effect of lines 5 and 9 is to obtain the next time slot at which to initiate a probe pair. Lines 15 and 17 increment the probe sequence and schedule the first probe of the pair, and lines 20 and 22 do the same for the second probe of the pair. Line 25 schedules the next time slot for this MADcode fragment to be executed. While Listing 3 implements the basic algorithm of BADABING (*i.e.*, probe pairs, rather than triple probes), implementing the improved algorithm is straightforward using MADcode.

Listing 1: MADcode for periodic probes used in delay experiments.

```
1  ; on entry:
2          ; r0 (read-only): current time slot
3          ; r1 (read-write): probe sequence
4
5  load r2, #interval
6          ; assign r2 literal value #interval
7          ; (the probe period, in time slots)
8  add r2, r0
9          ; r2 := r2 + current slot number
10 add r1, #1
11         ; increment probe sequence by 1
12 schedule_callback r2
13         ; schedule callback for slot r2
14 schedule_probe r2, r1
15         ; send probe at slot r2
16         ; with sequence r1
17
18 ; on exit:
19         ; probe sequence (r1) saved to user state
20         ; r2, r3 saved to user state
```

Listing 2: MADcode for geometrically distributed probes used in packet pair experiments.

```
1  ; on entry:
2          ; r0 (read-only): current time slot
3          ; r1 (read-write): probe sequence
4
5  geom r2, #numerator
6          ; store geometrically distributed
7          ; random number with parameter
8          ; p = #numerator/100 in r2
9  add r2, r0
10         ; r2 := r2 + current slot number
11 add r1, #1
12         ; increment probe sequence
13 schedule_callback r2
14         ; schedule callback for slot r2
15 schedule_probe r2, r1
16         ; send probe at slot r2
17         ; with sequence r1
18
19 ; on exit:
20         ; probe sequence (r1) saved to user state
21         ; r2, r3 saved to user state
```

Listing 3: MADcode for implementing BADABING geometrically distributed probe pairs used in loss experiments.

```
1  ; on entry:
2          ; r0 (read-only): current time slot
3          ; r1 (read-write): probe sequence
4
5  geom r2, #numerator
6          ; store geometrically distributed
7          ; random number with parameter
8          ; p = #numerator/100 in r2
9  add r2, r0
10         ; r2 := r2 + current slot number
11 load r3, r2
12         ; r3 := r2
13 add r3, #1
14         ; increment r3 by 1
15 add r1, #1
16         ; increment probe sequence by 1
17 schedule_probe r2, r1
18         ; schedule probe for slot r2
19         ; with sequence r1
20 add r1, #1
21         ; increment probe sequence
22 schedule_probe r3, r1
23         ; send probe at slot r3
24         ; with sequence r1
25 schedule_callback r3
26         ; schedule callback for slot r3
27
28 ; on exit:
29         ; probe sequence (r1) saved to user state
30         ; r2, r3 saved to user state
```

4.3 Implementation

There are three software components to MAD: the main MAD daemon, which receives, processes, and coordinates user requests to send probes, a probe reflector, and a probe receiver. (We use the term MAD to specifically refer to the probe sending daemon, unless otherwise stated.) A user accesses MAD through a remote procedure call interface. To initiate a new set of measurements, a user specifies up to 16 32-bit words of MADcode, along with the maximum number of probes to send, the destination IP address and port, the source address and port, the probe packet size, and the number of packets per probe (*e.g.*, BADABING specifies 3). If there is more than 1 packet per probe, the user also specifies what the spacing should be between those packets. The user must also present a 32-bit user-specific identifier, as described below. There are also RPC facilities for aborting an existing probe process, querying the number of probes sent for a particular user, and querying the MAD daemon for its internal statistics and error information.

One of the key enablers for MAD to run effectively on a standard, unmodified Linux system has been the introduction of scheduling and timer-related features from the real-time operating systems community into the mainline kernel code base. These additions originally appeared as patches to the 2.6.13 kernel, and were eventually incorporated into version 2.6.16 [5, 6]. One critical enhancement included modifications to add a high-resolution timer subsystem, providing timing capabilities finer than the default scheduler quantum (*i.e.*, the HZ parameter, typically set to 1000 in recent Linux kernels, giving a default timer granularity of 1 millisecond). The `nanosleep` system call uses this new timer subsystem, whereas the previous implementation (*e.g.*, versions previous to 2.6.16) performs a busy-wait.

MAD utilizes the new real-time features of recent Linux kernels, and in addition uses the real-time scheduling capabilities of Linux and runs at the highest possible priority level. Running as a high-priority process implies that MAD must run very efficiently or it may negatively impact system-wide performance. In the next section we evaluate MAD's impact on the host system and show that under typical use, it has extremely low resource requirements.

We implemented MAD in two ways: as a thread running from a kernel loadable module, and as a standalone user-mode process requiring privileged (root) access in order to run at a high priority. Both versions of MAD use a raw socket for sending UDP probe packets, to allow simple modification of source addresses for multi-homed hosts, and other protocol header elements. A compile-time parameter specifies the maximum number of users that can simultaneously use MAD. Another compile-time parameter specifies the maximum number of IP destinations that MAD can support. For each destination, MAD maintains a time-slot ordered list of probes to send in the future.

The in-kernel implementation uses kernel-equivalent calls to use standard the UDP/IP stack, and directly uses the high-resolution timer subsystem. In our experiments, described in the next section, we found that there was little qualitative performance difference between the in-kernel version, and the user-mode version. This result is good news, since it is better from a system reliability perspective to avoid implementation in kernel space where a programming bug can take down the entire system.

We also implemented the probe reflector as both a user-mode process and as an in-kernel daemon. As with the MAD daemon, we did not observe qualitative differences in performance. The implementation of the reflector is straightforward, and uses the BSD socket interface in a standard way to receive and send probes. One important run-time option is to rewrite the destination port or IP address with supplied values rather than simply swap the source and destination addresses and ports.

The implementation of the probe receiver is also straightforward. It is implemented only as a user-mode process. It is important to note that both probe reflector and receiver use the `SO_TIMESTAMP` option of `setsockopt` to obtain kernel-level timestamps. Currently, the receiver simply writes received probe information to disk. In the future, we plan to implement an RPC interface so that a user of MAD can both initiate a set of measurements, and retrieve the probes from the receiver host remotely.

All components of MAD are written in C, and comprise about 3,000 lines of code (including both user-mode and in-kernel versions of the MAD daemon and probe reflector). MAD will be made available to the research community.

In order to initiate a measurement process, a user must present a 32-bit identifier. Our intention is for this identifier to represent a time-limited access token to MAD, similar to a Kerberos ticket. Using traditional public key encryption, a user could be authenticated and given a token authorizing the user access to MAD for a limited duration. The token can be encrypted using less heavyweight private key encryption, with the key shared between the token process and the MAD daemon. This functionality is not yet implemented. In addition, accounting and quota facilities in MAD are not yet implemented.

There are certain limitations to MAD that we plan to address in the future. The first is that stream-based probe algorithms such as Pathload [21] and Yaz [41] are not yet easily implemented as MADcode. The reason is that the duration over which a stream is sent (for these algorithms 50–100 packets) can be longer than the discrete time interval used by MAD, *e.g.*, 5 milliseconds. It is not yet clear how to best enhance MAD to permit these types of algorithms. Adaptive algorithms are also not yet easily implemented, primarily because of the lack of a remote interface to the probe receiver. If a user can obtain measurements remotely from the probe receiver, it can then use that information for requesting a new set

of measurements to MAD, thus completing the feedback loop. Additionally, user code can only have one callback outstanding. This limitation exists partly for safety: allowing users to add an arbitrary number of callbacks is probably not necessary nor wise. However, some flexibility may be necessary for implementing some probe algorithms. Another limitation is that packet payloads cannot be modified by MAD users. It is not yet clear how to arbitrate among multiple users wishing to specify payload content, or whether this would be a useful feature. Finally, all users are treated at the same priority level in MAD. It may be useful in the future to be able to associate a priority level with a user's credentials in order to provide different levels of scheduling service quality.

5. EVALUATION

In this section we describe the evaluation of MAD in a controlled laboratory setting. We describe two different laboratory setups to examine the performance of MAD over a range of synthetic workloads on the measurement hosts and in both virtualized and non-virtualized operating system configurations. In each environment, we compare the performance of MAD with that of an unprivileged user-level process. Finally, we report results of microbenchmark scalability tests performed on MAD.

5.1 Controlled PlanetLab Experiments

In our first laboratory setup for evaluating MAD, we created a PlanetLab environment using the MyPLC software available at `http://www.planet-lab.org` [3]. We used version 0.4.3 of the MyPLC software. The node operating system included with this software is based on version 2.6.17 of the Linux kernel (a newer version than is running on current production PlanetLab hosts shown on CoMon [26]).

For our laboratory PlanetLab nodes we used two identical workstations with Pentium 4 processors running at 2 GHz, each with 1 GB RAM and Intel Pro/1000 network adapters. (As in our initial experiments, interrupt coalescence was disabled.) We used a network configuration similar to the one used for our production PlanetLab hosts (shown in Figure 1) for collecting ground truth packet traces with Endace DAG 4.3 GE cards.

To create synthetic load conditions on the PlanetLab nodes, we used the Harpoon network traffic generator [37], replicated over a number of experimental slices ranging between 0 and 100, as shown in Table 6, resulting in five different load scenarios. For each slice in all experiments in which at least one slice was configured, we ran one Harpoon process with 5 threads for producing self-similar TCP traffic, and one thread for producing constant bit-rate UDP traffic. The threads are configured to produce a relatively low average traffic volume of about 100 Kb/s. The UDP threads in our tests are configured to produce 10 Kb/s using small (40 byte) packets, resulting in about 24 packets per second per UDP thread. This configuration has the (intended) effect of artificially raising CPU utilization due to overheads related to timer maintainence.

The measurement algorithms we tested using MAD were the same as we used in the experiments with the production PlanetLab systems, described in Table 1. For each load scenario, we first ran each of the three measurement algorithms using MAD running as a privileged (root) user. We then ran each of the three measurement algorithms using an additional slice without any special privileges, similar to the way in which the measurement algorithms were run on the live PlanetLab hosts. Note that none of the slices configured in our testbed had any processor or bandwidth limitation placed on them.

We first look at results from the experiments using round-trip delay probes. Table 7 shows quantiles of the delay distribution for

6: Configurations and characteristics of laboratory-based PlanetLab experiments. For each slice, one Harpoon process was started with 5 threads for producing self-similar TCP traffic, and one thread for producing constant bit-rate UDP traffic. The average ratio between TCP and UDP traffic produced was 90:10.

Number of slices	Traffic Volume (each direction)	Average CPU Utilization
0	0	0%
1	100 Kb/s	5%
10	1 Mb/s	60%
50	5 Mb/s	95%
100	10 Mb/s	100%

a setup using a standard PlanetLab slice for sending the probes, and for a setup using MAD. Results for each of the five load scenarios are shown. From the table, we first notice that in the setup using a standard PlanetLab slice for sending probes, the only load scenario in which there is less than 1 millisecond delay at the 99th percentile is the one without any Harpoon processes. At a load of 50 slices the results appear qualitatively similar to those measured on the live PlanetLab hosts (*cf.* Table 2). For the scenario using 100 slices, the workstation is extremely overloaded, resulting in excessive round-trip delays. Most importantly in Table 7, we observe that for all load scenarios, the round-trip delays measured using MAD are on the order of 100 microseconds, even in the extreme scenario of 100 slices.

Results from experiments using BADABING in the laboratory-based PlanetLab setup are shown in Table 8. The table shows results for the setup using a standard PlanetLab slice for sending the probes, and for a setup using MAD, for each of the five load scenarios. From the table, we observe that in the standard PlanetLab slice setup there is no loss measured in the 0, 1, and 10 slice configuration, with a small amount of loss observed in the 50 slice scenario, and significant loss in the 100 slice scenario. As with our experiments using the live PlanetLab hosts, we used our DAG measurement systems to confirm that these losses were measurement errors. Finally, we see in Table 8 that for all load scenarios in which MAD is used, there is no loss measured by the BADABING probes.

For the packet pair experiments, we observed results that were similar to the experiments run on the live PlanetLab systems. In comparing the intended spacing between packets of a pair on send versus the actual produced, we found that the mean error is close to zero when considering a few tens of packet pairs. Similarly, the error mean between the spacing measured by the sending application and actual the spacing of the packet pair was close to zero for a few tens of packet pairs. This observation is true for both MAD and for a unprivileged process running within a standard slice. We found that as load increased, the interquartile range increased, *i.e.*, it spread further from the median in both the negative and positive error directions. The interquartile range was consistently larger for the unprivileged process, but not significantly so. For example, with the highest load scenario, the 25th percentile error was about -8 microseconds for the unprivileged process and about -6 microseconds for MAD, while the 75th percentile error was about 8 microseconds for the unprivileged process and about 5 microseconds for MAD. The median error was close to zero in both cases.

For errors in application-measured receive spacing versus the spacings measured using the DAG cards, we observed a similar pattern as in our live PlanetLab experiments, namely that of a shift away from zero mean error toward consistently positive error (*i.e.*, spacing measured at DAG was less than spacing measured in oper-

7: Quantiles of the delay distribution measured on laboratory PlanetLab nodes. Results shown for a standard PlanetLab slice (similar to the experiments using live PlanetLab nodes) and for MAD.

Number of	Standard PlanetLab slice				MAD			
Slices	50	90	95	99	50	90	95	99
0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000
10	0.000	0.001	0.001	0.002	0.000	0.000	0.000	0.000
50	0.001	0.002	0.002	0.049	0.000	0.000	0.000	0.000
100	1.750	15.291	21.801	29.329	0.000	0.000	0.000	0.000

8: BADABING results for laboratory PlanetLab nodes. Results shown for a standard PlanetLab slice (similar to the experiments using live PlanetLab nodes) and for MAD.

Number of	Standard PlanetLab slice			MAD		
Slices	Frequency	Duration	Rate Estimate	Frequency	Duration	Rate Estimate
0	0.0000	0.000	0.0000	0.0000	0.000	0.0000
1	0.0000	0.000	0.0000	0.0000	0.000	0.0000
10	0.0000	0.000	0.0000	0.0000	0.000	0.0000
50	0.0024	0.000	0.0024	0.0000	0.000	0.0000
100	0.0447	2.447	0.0447	0.0000	0.000	0.0000

ating system and application). At the same time, the interquartile range increased as load increased, similar to our observations in the case of send spacing timestamp error, though the relative increase was much larger. These observations hold for both MAD and for the unprivileged process running in a standard slice. For example, at the highest load scenario, the median error for the unprivileged process was about 16 microseconds, while the median error for MAD was about 10 microseconds. We measured similar results for MAD running both as a privileged user-mode process, and as a kernel module. These results suggest that running packet pair experiments on highly loaded hosts may be inherently problematic. It is unclear at this point whether the same problem exists for probe methodologies that use short streams at fixed rates, *e.g.*, Pathload [21] or Yaz [41]. At the very least, these results warrant further experimentation using a wider range of load scenarios, exploring both CPU and network-intensive workloads. Furthermore, it may be appropriate to examine a kernel-based approach that completely bypasses the system IP/UDP layers. Unfortunately, such an approach may require modifying the kernel.

In summary, our tests using PlanetLab in a controlled environment reveal similar measurement problems as we observed using the live PlanetLab systems. Our tests also show that using MAD significantly improves the situation and yields delay and loss measurements that accurately reflect the true state of the network. While our experiments show that MAD offers modest improvement for packet-pair experiments, additional study and improvements are needed.

5.2 Non-virtualized Host Experiments

In our second laboratory setup we used standard, unvirtualized workstations. Environments such as RON [10] form an important class of "raw system" testbeds for which MAD is designed. It is therefore important to evaluate the performance of MAD in a similar setting.

We used two workstations in our setup, similar to our other experiments. Each workstation ran Linux kernel version 2.6.20. We used identical machines as in our laboratory-based PlanetLab experiments. Each one had a 2.0 GHz Pentium 4, 1 GB RAM, and Intel Pro/1000 network interfaces (with interrupt coalescence disabled). We again used a setup using Endace DAG 4.3 GE cards to gather ground truth measurements, similar to our laboratory experiments using PlanetLab and in our experiments using live PlanetLab hosts.

For these tests, we again used Harpoon to create artificial CPU and network load on the hosts and testbed to compare the performance of an unprivileged user process for sending probes with MAD. Table 9 shows the four workload scenarios we used. As with the experiments using the controlled laboratory setup of PlanetLab, we configured the threads for generating constant bit-rate UDP traffic in such a way as to consume a relatively large amount of processor time.

9: Configurations and characteristics of laboratory experiments with non-virtualized hosts.

Harpoon configuration	Traffic Volume (each direction)	Average CPU Utilization
No Harpoon processes.	0	0%
10 Harpoon processes, each with 10 threads for producing self-similar TCP traffic and 1 thread for producing constant bit-rate UDP traffic.	2 Mb/s	2%
100 Harpoon processes, each with 10 threads for producing self-similar TCP traffic, and 5 threads for producing constant bit-rate UDP traffic.	10 Mb/s	60%
300 Harpoon processes, each with 10 threads for producing self-similar TCP traffic, and 20 threads for producing constant bit-rate UDP traffic.	50 Mb/s	99%

Using standard, unvirtualized hosts dramatically improves performance for the unprivileged user-mode application for all three measurement algorithms. In the case of round-trip delay, only the 99th percentile delay for the highest load scenario was above one millisecond—at two milliseconds. All other delay quantiles were below one millisecond. For packet loss, there was no loss measured in the lowest three load scenarios. However, in the highest load scenario the loss frequency was 0.0053 and the mean duration of loss episodes was 1.12 seconds.

For the experiments using MAD, the 99th percentile delay was on the order of one hundred microseconds in all cases. There was no loss measured by the MAD-based probes in any case.

For the packet pair experiments, the results were similar to those in the PlanetLab-based laboratory experiments. Namely, while errors in spacing of packet pairs upon sending, and errors in timestamping the packet pair upon send is relatively low with a zero mean over a few tens of packet pairs, timestamp errors on receiving packet pairs grow larger with increased system load. Also, the range in error values was similar to those in the controlled PlanetLab experiments. These results are true for both the unprivileged user-mode measurement application and for MAD. These results reinforce the hypothesis that running packet pair experiments on highly loaded hosts may be a situation to avoid entirely. But again, further experimentation and analysis is needed.

In summary, even with hosts that are not virtualized, measurement inaccuracies can occur as system load becomes high. Our experiments show that MAD is also effective in these environments, eliminating spurious packet loss and yielding delay estimates that match ground truth measurements.

5.3 Scalability of MAD

In our final laboratory experiments, we examined the scalability of MAD when subjected to a larger number of independent users of MAD, and over a range of discrete time interval settings. In these experiments, we used a standard (unvirtualized) Linux host running kernel version 2.6.20. The machine configuration used was the same in the other laboratory-based experiments, *i.e.*, a 2 GHz Pentium 4 with 1 GB RAM and an Intel Pro/1000 Gigabit Ethernet interface. The duration of these experiments was two minutes.

We used up to 100 independent probe streams representing 100 users of MAD, each using a geometric probe process (Listing 2) with a single packet per probe of 100 bytes. The probability parameter for sending a probe at a given time slot was 0.2 for each probe stream. We also ran experiments using the BADABING code fragment (Listing 3) with three packets per probe, sent back-to-back. The results from those experiments were similar to the results for the geometrically distributed probe consisting of one packet. We also examined MAD using a range of discrete time interval settings, from 10 milliseconds down to 100 microseconds. For each experiment, we measured system and user time available from the `getrusage` system call to derive a processor utilization figure for the MAD process. We also compared this utilization figure to that obtained using the standard `top` program. The results for each measurement technique were consistent.

Table 10 shows CPU utilization results of running MAD with 100 independent probe streams over a range of discrete time intervals. From the table, we see that for time intervals even as short as 500 microseconds, the overall utilization of MAD is minor, at about 0.2%. At intervals of 500 microseconds and larger, there were no scheduler errors reported. At an interval of 100 microseconds, however, utilization rises sharply and is accompanied by scheduler errors. With the default setting of 5 milliseconds (which is the same interval used in our earlier BADABING study) and even shorter intervals, MAD performs very well.

10: Results from MAD scalability tests.

Interval	CPU Utilization	Scheduler Errors (time slot misses)
10 milliseconds	0.1%	0
5 milliseconds	0.1%	0
1 millisecond	0.2%	0
500 microseconds	0.2%	0
100 microseconds	46.2%	8920

6. SUMMARY AND CONCLUSIONS

Widely deployed, shared network testbeds are critical to the network research community. A particularly attractive class of experiments that could be considered in these environments are those that seek to measure end-to-end path properties such as delay and loss using active probe tools. Unfortunately, the resource contention overheads imposed in shared network testbeds can significantly bias measurement results from active probe tools—a compelling yet unfortunate example of the "tragedy of the commons" effect.

In this paper we present results of a measurement study that quantifies the bias effects on active probe-based measurements in PlanetLab. Using hardware-based packet capture systems on our local PlanetLab nodes, we find that measurements of packet loss and delay from active probes can be skewed significantly.

These results motivate our development of MAD, a system for conducting highly accurate active measurements in shared environments. MAD is realized as a simple programming language that is made available to users via RPC's. The language enables a variety of active probe-base measurement streams to be scheduled in near real-time through the use of priority scheduling mechanisms available in recent Linux kernels. MAD'S implementation as either a kernel module or user-mode daemon enables it to be deployed with minimal impact. Through a series of laboratory tests, we quantify the extent to which MAD can reduce bias in active probe-based measurements in both PlanetLab and in non-virtualized environments. We show that mad can improve measurement accuracy by orders of magnitude in PlanetLab with lesser but still valuable effects in non-virtualized environments.

We plan to continue development of MAD in several ways. First, we intend to complete the implementation of the security mechanism that limits access of MAD to authorized users. Next, we plan to expand MADcode to support of a broader set of active measurement methods including those that are stream-based or adaptive. Finally, we will consider how basic MAD transmitter/receiver/reflector functionality might be ported to other OS environments so that highly accurate measurement capability might be more widely deployed.

Acknowledgments

We thank our shepherd, Morley Mao, and the anonymous IMC reviewers for their input. We also thank Mike Blodgett for his assistance with the DAG systems. This work was supported in part by NSF grants CNS-0347252, CNS-0646256, CNS-0627102, and by Cisco Systems. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or Cisco Systems.

7. REFERENCES

[1] EverLab: Next Generation PlanetLab Network. `http://www.everlab.org`.
[2] KURT: Kansas University Real-time Linux. `http://www.ittc.ku.edu/kurt/`.
[3] MyPLC—A complete Planetlab Central (PLC) portable installation. `http://www.planet-lab.org/doc/myplc`.
[4] OneLab. `http://www.fp6-ist-onelab.eu/`.
[5] A new approach to kernel timers. `http://lwn.net/Articles/152436/`, September 2005.

[6] Linux kernel gains new real-time support. http://www.linuxdevices.com/news/-NS9566944929.html, October 2006.

[7] NSF CISE, GENI — Global Environment for Network Innovations . http://www.geni.net, 2007.

[8] G. Almes, S. Kalidindi, and M. Zekauskas. A one-way delay metric for IPPM. IETF RFC 2679, September 1999.

[9] G. Almes, S. Kalidindi, and M. Zekauskas. A one way packet loss metric for IPPM. IETF RFC 2680, September 1999.

[10] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of ACM Symposium on Operating Systems Principles*, Banff, Alberta, Canada, 2001.

[11] M. Aron and P. Druschel. Soft Timers: Efficient Microsecond Software Timer Support for Network Processing. *ACM Transactions on Computer Systems*, August 2000.

[12] S. Banerjee, T. Griffin, and M. Pias. The interdomain connectivity of PlanetLab nodes. In *Proceedings of Passive and Active Measurement Workshop*, Antibes Juan-les-Pins, France, April 2004.

[13] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Heugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of ACM Symposium on Operating Systems Principles*, October 2003.

[14] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Network Services. In *USENIX Symposium on Networked Systems Design and Implementation*, March 2004.

[15] B. Bershad, S. Savage, P. Pardyak, E. Sirer, M. Fiuczynski, D. Becker, S. Eggers, and C. Chambers. Extensibility, safety and performance in the SPIN operating system. In *Proceedings of ACM Symposium on Operating Systems Principles*, Copper Mountain Resort, CO, December 1995.

[16] J. Bolot. End-to-end packet delay and loss behavior in the internet. In *Proceedings of ACM SIGCOMM*, San Francisco, CA, September 1993.

[17] R. Carter and M. Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance Evaluation Review*, 27-28:297–318, October 1996.

[18] National Research Council, editor. *Looking Over the Fences at Networks: A Neighbor's View of Networking Research*. National Academy Press, 2001.

[19] D. Engler and M. Kaashoek. Exokernel: an operating system architecture for application-level resource management. In *Proceedings of ACM Symposium on Operating Systems Principles*, Copper Mountain Resort, CO, December 1995.

[20] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM*, Stanford, CA, 1988.

[21] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation to TCP throughput. In *Proceedings of ACM SIGCOMM*, Pittsburgh, PA, August 2002.

[22] S. Kalidindi and M. Zekauskas. Surveyor: An Infrastructure for Internet Performance Measurements. In *Proceedings of INET '99*, 1999.

[23] R. Kapoor, L.-J. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi. CapProbe: a simple and accurate capacity estimation technique. In *Proceedings of ACM SIGCOMM*, Portland, OR, August 2004.

[24] K. Lai and M. Baker. Measuring link bandwidths using a deterministic model of packet delay. In *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, 2000 2000.

[25] J. Liu and M. Crovella. Using loss pairs to discover network properties. In *Proceedings of ACM Internet Measurement Workshop*, San Francisco, CA, October 2001.

[26] K. Park and V. Pai. CoMon—A Monitoring Infrastructure for PlanetLab. http://comon.cs.princeton.edu/.

[27] A. Pásztor and D. Veitch. A precision infrastructure for active probing. In *Proceedings of Passive and Active Measurement Workshop*, Amsterdam, Netherlands, 2001.

[28] A. Pásztor and D. Veitch. PC-based Precision Timing without GPS. In *Proceedings of ACM SIGMETRICS*, Marina Del Rey, CA, June 2002.

[29] V. Paxson, A. Adams, and M. Mathis. Experiences with NIMI. In *Proceedings of Passive and Active Measurement Workshop*, 2000.

[30] L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet Impasse through Virtualization. In *Proceedings of ACM SIGCOMM HotNets–III*, 2004.

[31] H. Pucha, Y.C. Hu, and Z.M. Mao. On the Impact of Research Network based Testbeds on Wide-Area Experiments. In *Proceedings of ACM Internet Measurement Conference*, Rio de Janeiro, Brazil, October 2006.

[32] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathChirp: Efficent Available Bandwidth Estimation for Network Paths. In *Proceedings of Passive and Active Measurement Workshop*, April 2003.

[33] M. Rosenblum and T. Garfinkel. Virtual Machine Monitors: Current Technology and Future Trends. *IEEE Computer*, May 2005.

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

[35] E. Sarmiento. Securing FreeBSD using Jail. *Sys Admin*, 10(5):31–37, May 2001.

[36] S. Soltesz, H. Pötzl, M. Fiuczynski, A. Bavier, and L. Peterson. Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors. In *Proceedings of EuroSYS*, 2007.

[37] J. Sommers and P. Barford. Self-configuring network traffic generation. In *Proceedings of ACM Internet Measurement Conference*, Taormina, Sicily, Italy, October 2004.

[38] J. Sommers, P. Barford, N. Duffield, and A. Ron. Improving Accuracy in End-to-end Packet Loss Measurement. In *Proceedings of ACM SIGCOMM*, Philadelphia, PA, August 2005.

[39] J. Sommers, P. Barford, N. Duffield, and A. Ron. A Framework for Multi-objective SLA Compliance Monitoring. In *Proceedings of IEEE INFOCOM (minisymposium)*, Anchorage, AK, May 2007.

[40] J. Sommers, P. Barford, N. Duffield, and A. Ron. Accurate and Efficient SLA Compliance Monitoring. In *To appear, Proceedings of ACM SIGCOMM*, Kyoto, Japan, August 2007.

[41] J. Sommers, P. Barford, and W. Willinger. A proposed framework for calibration of available bandwidth estimation tools. In *Proceedings of IEEE Symposium on Computer and Communication*, Pula, Sardinia, Italy, June 2006.

[42] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using PlanetLab for Network Research: Myths, Realities, and Best Practices. In *Proceedings of the Second USENIX Workshop on Real, Large Distributed Systems (WORLDS '05)*, San Francisco, CA, December 2005.

[43] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A Public Internet Measurement Facility . In *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[44] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of ACM Internet Measurement Conference*, Miami, FL, October 2003.

[45] Y. Zhang, R. West, and X. Qi. A virtual deadline scheduler for window-constrained service guarantees. In *Proceedings of the 25th IEEE Real-time Systems Symposium (RTSS)*, December 2004.