Using PlanetLab for Network Research: Myths, Realities, and Best Practices

Neil Spring, Larry Peterson, Andy Bavier, and Vivek Pai

1 Introduction

PlanetLab is a research testbed that supports 428 experiments on 276 sites, with 583 nodes in 30 countries. It has lowered the barrier to distributed experimentation in network measurement, peer-to-peer networks, content distribution, resource management, authentication, distributed file systems, and many other areas.

PlanetLab did not become a useful network testbed overnight. It started as little more than a group of Linux machines with a common password file, which scaled poorly and suffered under load. However, PlanetLab was conceived as an *evolvable* system under the direction of a community of researchers. With their help, PlanetLab version 3.0 has since corrected many previous faults through virtualization and substantial performance isolation. This paper is meant to guide those considering developing a network service or experiment on PlanetLab by separating widely-held myths from the realities of service and experiment deployment.

Building and maintaining a testbed for the research community taught us lessons that may shape its continued evolution and may generalize beyond PlanetLab to other systems. First, users do not always search out "best practice" approaches: they expect the straightforward approach to work. Second, users rarely report failed attempts: we learned of the perceived shortcomings described in this paper through conversations, not through messages to the mailing lists. Third, frustration lingers: users hesitate to give another chance to a system that was recently inadequate or difficult to use. These experiences are especially challenging for an evolvable system, which relies on user feedback to evolve so that more users can be supported by features they desire.

We organize the myths in decreasing order of veracity: those that are realities in Section 2, that were once true in Section 3, and those that are false if best practices are employed in Section 4. We summarize the discussion in Section 5.

2 Realities

This section describes widely-cited criticisms of PlanetLab that are entirely true, and are likely to remain so even as PlanetLab evolves.

Reality: Results are not reproducible

PlanetLab was designed to subject network services to real-world conditions, not to provide a controlled environment. By running a service for months or years, researchers should be able to identify trends and understand the performance and reliability their service achieves. An experiment that runs for an hour will reflect only the conditions of the network (and PlanetLab) during that hour.

Various aspects of a service can be meaningfully measured by applying simple rules-of-thumb. Avoid heavily-loaded times and nodes: CoMon [5] tracks and publishes current resource usage on each PlanetLab node. Secure more resources for your experiment from a brokerage service (see Section 3) if needed. Repeat experiments to generate statistically valid results. Finally, regard PlanetLab's ability to exercise a system in unintended ways, producing unexpected results, as a feature, not a bug.

Reality: The network between PlanetLab sites does not represent the Internet

No testbed, no simulator [2], and no emulator is inherently representative of the Internet. The challenges for researchers are to develop experiments that overcome this limitation, perhaps by recruiting real users behind residential access networks, or, failing that, to interpret results taking PlanetLab's special network into account. The challenge for PlanetLab is to evolve so that this limitation is less severe, seeking new sites and new access links.

PlanetLab's network is dominated by global research and education network (GREN) [1] (Internet2 in the United States). However, commercial sites have joined PlanetLab and research sites have connected machines to DSL and cable modem links: 26 sites are purely on the commercial Internet. The question is, how does PlanetLab's network connectivity affect research?

First, some experiments are suitable for the GREN. Claims that a new routing technique can find better routes than BGP are suspect if those better routes take advantage of well-provisioned research networks that are not allowed by BGP policy. However, claims that a service can find the best available route might be accurate even on the GREN: results obtained on the GREN are not necessarily tainted.

Second, services for off-PlanetLab users and network measurement projects that send probes off-PlanetLab observe the commercial Internet. Although most of PlanetLab is on the GREN, most machines also connect to the commercial network or are part of transit ASes. The PlanetFlow auditing service [4] reports that PlanetLab nodes communicate with an average of 565,000 unique IP addresses each day. PlanetSeer [10], which monitors TCP connections between CoDeeN nodes at PlanetLab sites and Web clients/servers throughout the Internet, observed traffic traversing 10,090 ASes, including all tier-1 ISPs, 96% of the tier-2 ISPs, roughly 80% of the tier-3 and 4 ISPs, and even 43% of the tier-5 ISPs. Measurement services like Scriptroute [7] can use the geographic diversity of vantage points provided by PlanetLab to probe the Internet without being limited by the network topology between PlanetLab nodes.

Finally, it is sometimes not the topology of the GREN, but the availability of its very high bandwidths and low contention that calls results into question. Researchers can, however, limit the bandwidth their slices consume to emulate a lower bandwidth link, via user-space mechanisms (e.g., pacing the send rate) or by asking PlanetLab support to lower the slice's outgoing bandwidth cap.

Reality: PlanetLab nodes are not representative of peer-to-peer network nodes

Typically, this is a comment about the high-bandwidth network (see above). Sometimes it means that PlanetLab is a managed infrastructure and not subject to the same churn as desktop systems.

Although PlanetLab is not equivalent to a set of desktop machines—and it is not expected to scale to millions of machines—it can contribute to P2P services. A "seed deployment" on PlanetLab would show the value of a new service and encourage end-users to load the service on desktop machines. End System Multicast [3] instead uses PlanetLab nodes as the "super nodes" of a P2P network. PlanetLab can contribute a core of stable, managed nodes to P2P systems.

3 Myths that are no longer true

Some who tried to use early versions of PlanetLab found challenges that are no longer so daunting because PlanetLab has evolved.

Myth: PlanetLab is too heavily loaded

Although PlanetLab may always be under-provisioned and load is especially high before conference deadlines, this perception is misleading in two ways.

First, upgrades to the OS better tolerate high CPU load, memory consumption, and disk access load. CPU cycles are fairly distributed among slices rather than threads: a slice with 100 threads receives *the same* CPU allocation as a slice with just one. A daemon polices memory consumption, killing slices that use too much when memory pressure is high; users now take greater care in configuring programs that may have a heavy memory footprint to avoid having them killed, which in turn has reduced memory pressure for everyone. Finally, an OS upgrade enables disk access via DMA, rather than programmed I/O, improving performance when the node is swapping.

Second, PlanetLab has two brokerage services, Sirius and Bellagio, that perform admission control to a pool of resources. Researchers can use these services to receive more than a "fair share" of the CPU, for fixed periods of time, during periods of heavy load.

CPU availability measurements. An experiment begun in February 2005 supports the claim that PlanetLab has sufficient CPU capacity. The experiment runs a spin-loop on each PlanetLab node to sample the CPU available to a slice; because of PlanetLab's fair share CPU scheduler, this measurement is more accurate than standard techniques such as the load metric reported by `top`. Figure 1 summarizes seven months of CPU availability measurements. The three lines are the median, 25^{th}, and 10^{th} percentiles of the available CPU across all nodes. The median line shows that most nodes had at least 20% available: a slice on a typical PlanetLab node contends with three to five other slices that are running processes nonstop. The 25^{th} percentile line generally stays above 10%, indicating that fewer than one-fourth of the nodes had less than 10% free. A slice can get nearly 10% of the CPU on almost any node.

CPU time is also available immediately before conference deadlines as well. For example, during the week before the SIGCOMM deadline (February 1–8, 2005), 360 of the 362 running nodes (99%) had at least 10% available CPU, averaged over the week; 328 of the 360 nodes (91%) had at least 20% available. These results show somewhat higher availability than in Figure 1. Some projects may have refrained from using PlanetLab to leave resources available to those running last-minute experiments.

Estimates of available CPU using other metrics are less accurate. In Figure 2, we show the median capacities (a) measured directly using spin loops, (b) estimated using the inverse of the load average (a load of 100 equals 1% CPU availability), and (c) estimated using the inverse of the number of active slices (meaning slices with a runnable thread). The top line, the spin-loop measured capacity, is significantly higher. The Unix-reported load average is often misleading: the processors did have high load (sometimes exceeding 100), but the CPU available to slices is much greater because although slices that spawn many processes increase the load average, their processes compete only against each other for CPU. Likewise, not all active slices use their entire quanta and so the active slice count overestimates contention. The CoMon monitoring service now publishes the results of the spin-loop tests to help users choose nodes by CPU availability.

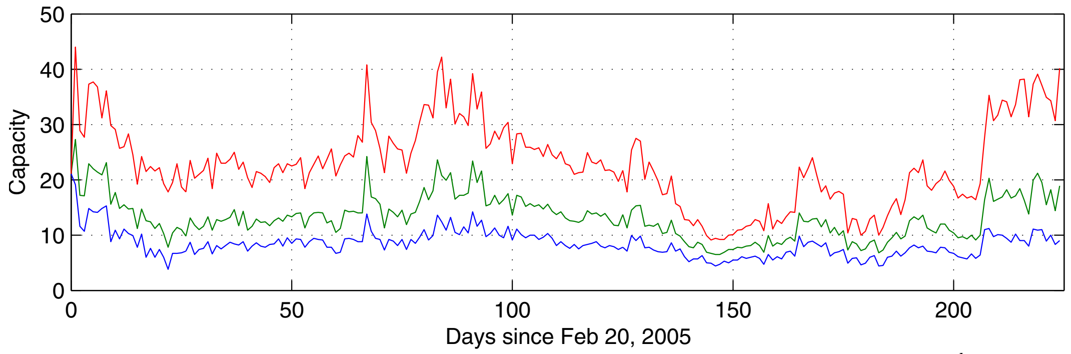

Figure 1: Available CPU across PlanetLab nodes. Median percentage available CPU is red (upper), 25th percentile is green (middle), and 10th is blue (lower).

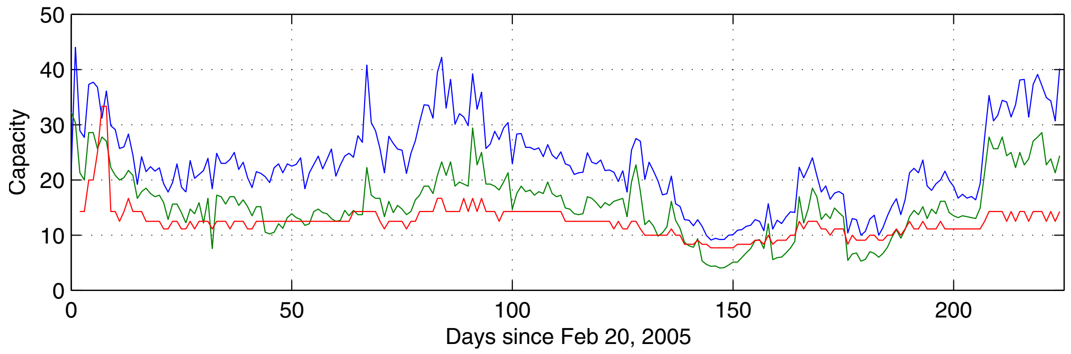

Figure 2: Median available CPU measurements using spin loops (blue, upper), load average (green, middle), and number of active slices (red, typically lowest).

Myth: PlanetLab cannot guarantee resources

Resource guarantees could not be given before version 3.0. Schedulers are now available to make resource guarantees, but PlanetLab does not yet have a policy about what slices should receive them. Typically, continuously running services on PlanetLab are robust to varying resource availability (and have not asked for guarantees), while short-term experiments have the option of using one of the brokerage services (see previous item) to gain sufficient capacity for the duration of a run. Once we have enough experience to understand what policies should be associated with guarantees, or someone develops a robust market in which users can acquire resources, resource guarantees are likely to become commonplace.

4 Myths falsified by best practices

The following four myths about PlanetLab are not true if best practices are followed. Often these myths are caused by mismatches between the behavior of a single, unloaded Linux workstation, and the behavior of a highly-shared, network of PlanetLab-modified Linux nodes. The first three myths address problems using PlanetLab for network measurement, the last, its potential for churn.

Myth: Load prevents accurate latency measurement

Because PlanetLab machines are loaded, no application can expect that a call to `gettimeofday()` right after `recv()` will return the time when the packet was received by the machine. The PlanetLab kernel scheduler (Section 3) can isolate slices so that none are starved of CPU, but cannot ensure that any slice will be scheduled immediately upon receiving a packet.

Using in-kernel timestamping features of Linux, network delay can be isolated from (most) processing delay. When a machine receives a packet, the network device sends an interrupt to the processor so that the kernel can pull the packet from the device's queue. At the point when Linux accepts the packet from the device driver, it annotates the buffer with the current time.[1] The kernel will return control to the current process for the remainder of its quantum, but this timestamp is kept in the kernel and made available in at least three ways:

1. The SIOCGSTAMP ioctl called after reading a packet. Ping uses this ioctl, but Linux kernel comments suggest the call is Linux-specific.
2. The SO_TIMESTAMP socket option combined with `recvmsg()`: ancillary data includes a timestamp. The Spruce [8] receiver code uses this method, which was introduced in BSD and is supported by Linux. It is not widely documented, but can be run as a non-root user.
3. The library behind tcpdump, libpcap. This may be the most portable, but requires root, which is easy on PlanetLab. *Sent* packets are also timestamped [9].

[1] See: linux/net/core/dev.c:netif_rx().

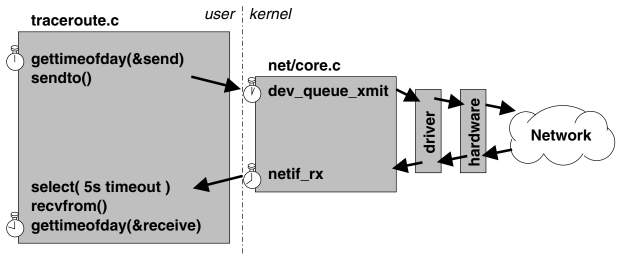

Figure 3: Approaches to packet round-trip timing: applications can use gettimeofday before sending and after receiving; closer to the device are kernel-supplied timestamps applied as the packet is queued for transmission or received. The driver and hardware also may delay packets on transmission and receipt.

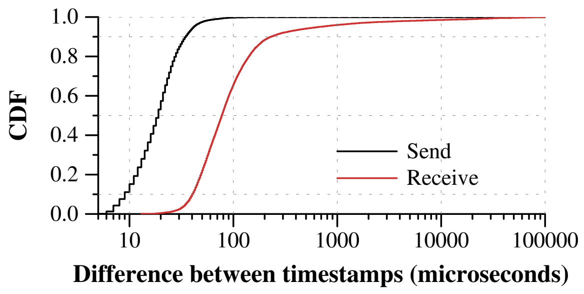

Figure 4: A cumulative distribution of the differences between application-level timestamps and kernel-level timestamps when sending (left) or receiving (right) in microseconds.

Do kernel timestamps matter? To collect samples of application- and kernel-level timestamps, we modified traceroute to print the timestamps it collects via `gettimeofday()`, then ran traceroute and tcpdump in parallel to gain kernel-level timestamps for the same packets from 300 PlanetLab machines to three destinations, collecting 40,000 samples for comparison. Figure 3 illustrates where traceroute and the kernel annotate timestamps.

In Figure 4, we show the differences between application- and kernel-captured timestamps when sending probes and receiving responses. Although the time between `gettimeofday()` and when the packet is delivered to the network device is typically small (18 μs median, 84 μs mean), the time after the packet is received is typically larger and more variable (77 μs median, 788 μs mean). The larger median may represent the cost of the intermediate system calls: in traceroute, it is `select()` that returns when the response packet is received. However, that 4% of samples are above 1 ms suggests contention with other active processes. Further, the smallest 3% of samples between 20–30 μs suggests that tools that filter for the minimum round trip time, such as pathchar, will have difficulty: 97% of the packets will not observe minimal delay in receive processing.

Measurement tools downloaded from research Web pages may not use kernel-level techniques to measure packet timings; their results should be held with skepticism until their methods are understood.

Myth: Load prevents sending precise packet trains

Sending packets at precise times, as needed by several tools that measure available bandwidth, is more difficult. If the process is willing to discard measurements where the desired sending times were not achieved or when control of the processor is lost, then sending rate-paced data on PlanetLab simply requires more attempts than on unloaded systems.

To determine how CPU load impairs precise sending, we measure how often we can send precisely-spaced packets in a train. Sent trains consist of eleven packets, spaced either by 1 ms, to test spin-waiting, or 11 ms, to test sleep-based waiting using the `nanosleep()` system call (via the `usleep()` library call). We show how often the desired gaps were achieved for 1 ms gaps in Figure 5 and 11 ms gaps in Figure 6. In all measurements, 10 gaps are used, and we measure how often the gaps are within 3% of the target either for all 10 gaps or for any 5 consecutive gaps.

For both tests, at least five consecutive gaps have the desired intervals in 80–90% of the trains. For the 11 ms test, all 10 gaps had the correct timing 60–70% of the time. The 1 ms test did not fare as well: all 10 gaps met their target times in only 20–40% of the trains. For the shorter (5-gap) chirp trains, the results are quite good: sending 10 packets is sufficient to discard less than 20% of the measurements. For longer chirp trains, two to five times as many probes may have to be sent, which may be tolerable for many experiments.

Mechanisms for negotiating temporarily longer time slices, or even delegating packet transmission scheduling to the kernel, are being discussed. The latter might address another source of concern for measurement experiments: the packet scheduler used to cap bandwidth and fairly share bandwidth among slices. The timestamps on sent packets that a process can observe with libpcap are accurate—the kernel timestamps packets *after* they pass through the packet scheduler—and so can still be used to discard bad results. However, the scheduler does limit the kinds of trains that can be sent: it enforces a per-slice cap of 10 Mbps with a maximum burst size of 30KB. Longer trains sent at a faster rate are not permitted.

Myth: The PlanetLab AUP makes it unsuitable for measurement

The PlanetLab user Acceptable Use Policy [6] states:

> PlanetLab is designed to support network measurement experiments that purposely probe the Internet. However, we expect all users to adhere to widely-accepted standards of network etiquette in an effort to minimize complaints from network administrators. Activities that have been interpreted as worm and denial-of-service attacks in the past (and should be avoided) include sending SYN packets to port 80 on random machines, probing random IP addresses, repeatedly pinging routers, overloading bottleneck links

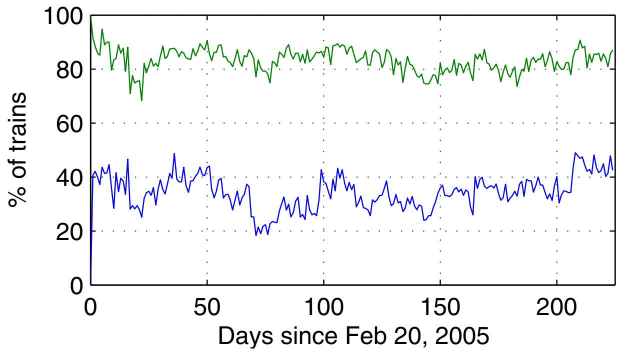

Figure 5: Timing statistics for 1 ms (spin-based) chirp trains. The green (upper) line indicates at least 5 consecutive gaps met the target timings, while the blue (lower) line indicates all gaps met the target.

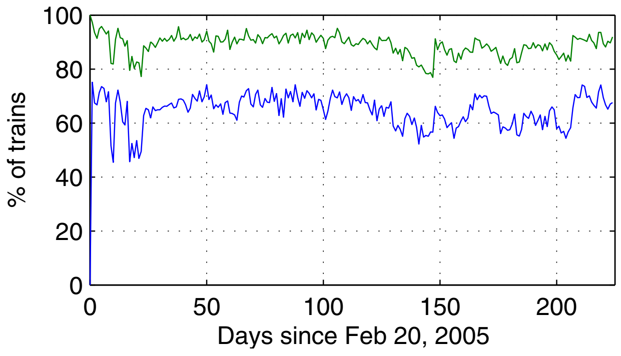

Figure 6: Timing statistics for 11 ms (sleep-based) chirp trains. The green (upper) line indicates at least 5 consecutive gaps met the target timings, while the blue (lower) line indicates all gaps met the target.

with measurement traffic, and probing a single target machine from many PlanetLab nodes.

This policy is a result of experience with network measurements on PlanetLab, and is designed to prevent network abuse reports of the form "PlanetLab is attacking my machine." Here we elaborate on steps to conduct responsible Internet measurement on PlanetLab. The goal of these practices is to make network measurements as easy to support as possible by building a list of hosts that "opt-out" of measurement without growing the list of PlanetLab sites that have asked to "opt-out" of hosting measurement experiments.

Test locally and start slow. Do not use PlanetLab to send traffic you would not send from your workstation. Use a machine at your site first to discover any problems with your tool before causing network-wide disruption. Measurements from PlanetLab can appear to be a distributed denial of service attack; starting with a few nodes can limit how many sites receive abuse reports. Some intrusion detection systems generate automatic abuse reports; an abuse report to every PlanetLab host is best avoided.

Software has bugs, and bugs can cause measurements to be more intrusive than necessary. Bugs that have made PlanetLab-supported tools unnecessarily intrusive include faulty checksum computation in a lightweight traceroute implementation and a reaction to unreachable hosts that directed a great deal of redundant measurement toward the same router. Such errors could have been detected before deployment with local testing.

Even a correctly-implemented tool may require local testing, because very little experimental data guides non-intrusive measurement tool design: are TCP ACKs less likely to raise alarms than SYNs? Should traceroute not increment the UDP destination port to avoid appearing as a port scan? How many probes are needed to distinguish lossy links from unreachable hosts?

Starting slow could have avoided abuse report flurries in March and October 2005. An experiment with an implementation flaw generated 19 abuse reports from as many sites, half on the first day, March 15. The experiment ran for only 21 hours before being shut down, but reports continued in for two weeks. A carefully-designed experiment in October tickled two remote firewalls and a local intrusion detection system for a total of 10 abuse reports forwarded to PlanetLab support. The automated responses from remote firewalls may have been avoided by local testing of the destination address list. Many more abuse reports were likely generated by the automated systems, but discarded by recipients as frivolous as they reported a single ICMP echo request (ping) as an attack.

Alert PlanetLab support. Update your slice description *and* send a message to PlanetLab support detailing your intended measurement, how to identify its traffic, and what you've done to try to avoid problems. First, sending such a message shows that you, as an experimenter, believe you have put sufficient effort into avoiding abuse reports. Second, describing your approach gives PlanetLab staff and other interested people the chance to comment upon your design. Finally, knowing the research goals and methods can save PlanetLab staff time and ensures prompt response to abuse reports.

Use Scriptroute. Scriptroute separates measurement logic from low-level details of measurement execution. It will prevent contacting hosts that have complained about traffic, can prevent inadvertently invalid packets that trigger intrusion detection systems, will limit the rate of traffic sent, collects timestamps from libpcap, and schedules probes using a hybrid between sleeping and busy-waiting.

Curtail ambition. It is tempting to demonstrate implementation skill by running a measurement study from *everywhere* to *everywhere*, using many packets for accuracy, and using TCP SYN packets to increase the chance of discovering properties of networks behind firewalls. Resist! Aggressive measurement increases its cost for only a marginal benefit to the authority of your result.

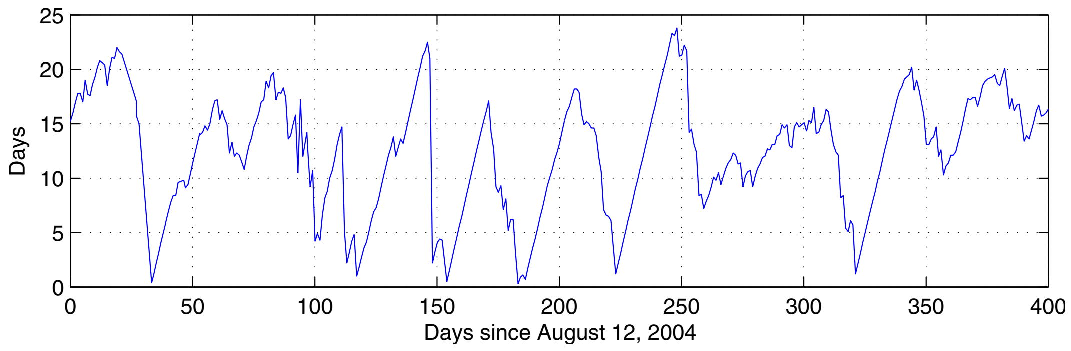

Figure 7: Median uptime in days across all PlanetLab nodes.

Myth: PlanetLab experiences excessive churn

Widespread outages on PlanetLab are fairly rare. Only three times during the last two years have many PlanetLab nodes been down for longer than a reboot: (1) all nodes were taken off-line for a week in response to a security incident in December 2003; the system was also upgraded from version 1.0 to 2.0; (2) an upgrade from version 2.0 to 3.0 during November 2004 caused more churn than usual for a two week period; and (3) a kernel bug in February 2005 took many nodes off-line for a weekend.

On the other hand, roughly 30% of PlanetLab's nodes are down at any given time. About one-third of these are down for several weeks, usually because a site is upgrading the hardware or blocking access due to an AUP or security issue. The remaining failed nodes are part of the daily churn that typically sees 15–20 nodes fail and as many recover each day. Major software upgrades that require reboots of all nodes occur, but are infrequent.

PlanetLab as a whole has been remarkably stable. Figure 7 shows median node uptimes over 13 months. Of the six sharp drops in uptime, four are due to testbed-wide software upgrades requiring reboots. The longer upgrade, to version 3.0, is shown starting at day 100. The kernel bug, followed by an upgrade, is evident starting at day 170. Median uptimes are generally longer than 5 days, and often 15 to 20 days—much higher than what would be expected in typical home systems.

Since PlanetLab does experience churn, no users should expect that the storage offered by PlanetLab nodes is persistent and no users should expect that a set of machines, once chosen, will remain operational for the duration of a long-running experiment.

5 Summary

In this paper, we described realities of the PlanetLab platform: it is not representative of the Internet or of peer-to-peer networks, and results are not always reproducible. We then described myths that linger despite being fixed: PlanetLab's notoriously high load poses less of a problem today than it once did because there are resource brokerage services and the operating system has been upgraded to isolate experiments. Finally, we described challenges that can often be addressed by following some best practices. PlanetLab is capable of substantial network measurement, despite technical challenges in precise timing and social challenges in avoiding abuse complaints. In addition, many PlanetLab machines may fail or be down at any time; being prepared for this churn is a challenge for experimenters.

Our hope is that separating myth from reality will make clear the features and flaws of PlanetLab as an evolving research platform, enabling researchers to choose the right platform for their experiments and warning them of the challenges PlanetLab implies.

Acknowledgments

We would like to thank the anonymous reviewers for their useful feedback on the paper. This work was supported in part by NSF Grants ANI-0335214, CNS-0439842, and CNS-0435065.

References

[1] S. Banerjee, T. G. Griffin, and M. Pias. The interdomain connectivity of PlanetLab nodes. In *PAM*, 2004.

[2] S. Floyd and V. Paxson. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, 2001.

[3] Y. hua Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS*, 2000.

[4] M. Huang, A. Bavier, and L. Peterson. PlanetFlow: Maintaining Accountability for Network Services. Submitted for publication.

[5] K. Park and V. Pai. CoMon: A monitoring infrastructure for PlanetLab. http://comon.cs.princeton.edu.

[6] PlanetLab Consortium. PlanetLab acceptable use policy (AUP). https://www.planet-lab.org/php/aup/PlanetLab_AUP.pdf, 2004.

[7] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A public Internet measurement facility. In *USITS*, 2003.

[8] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *IMC*, 2003.

[9] TCPDUMP.org Frequently Asked Questions. http://www.tcpdump.org/faq.html, 2001.

[10] M. Zhang, *et al.* PlanetSeer: Internet path failure monitoring and characterization in wide-area services. In *OSDI*, 2004.