Flexlab: A Realistic, Controlled, and Friendly Environment for Evaluating Networked Systems

Jonathon Duerig Robert Ricci Junxing Zhang Daniel Gebhardt Sneha Kasera Jay Lepreau

University of Utah, School of Computing

Abstract

Research prototypes of networked systems are often evaluated on overlay testbeds and emulation testbeds. Most of the strengths and weaknesses of these two types of testbeds are complementary. We outline the motivation, design, implementation, and sample results of an environment that seeks to provides the best of each type. Flexlab couples an emulation testbed with arbitrary network models. We also present a novel modeling technique tuned for this environment, *application-centric Internet modeling.* The key idea is to monitor the application's offered network load within the emulation testbed, replicate that load on the overlay testbed, measure the path's characteristics through analysis of the traffic, and use those to shape the emulated network.

1 Introduction

Emulation testbeds such as Emulab [25] and ModelNet [22] are valuable tools for understanding, testing and evaluating research prototypes of networked systems. They give users great control over host and network environments and offer easy reproducibility. However, emulation testbeds have a serious shortcoming: their network conditions are artificial and thus do not exhibit some aspects of real networks. Perhaps worse, researchers are *not sure* of two things: which network aspects are poorly modeled, and which matter to their application. We believe these are two of the reasons researchers underuse emulation environments; that emulators are underused has also been observed by others [23].

In this paper, we address this shortcoming by presenting Flexlab, a new testbed environment that enables a new generation of network models for emulation. We present three network emulation models built with Flexlab; they gather Internet measurements using PlanetLab. The first two use traditional measurement strategies, while application-centric Internet modeling is a novel technique for high-fidelity emulation. We rely on Emulab facilities to provide a friendly and controllable environment, but our techniques generalize to any emulator.

Currently, to get network conditions more realistic than those in an emulator, researchers use overlay testbeds such as PlanetLab [15] and RON [2], which provide a set of vantage points into the Internet. These testbeds also provide other orthogonal advantages: true geographic distribution, a service platform, and potential for deployment and real end-users. In this paper, however, we concentrate on their use as sources of realistic end-to-end network paths.

These live-network testbeds have some drawbacks that are not present in emulation testbeds. First, because of the popularity of overlay testbeds and the limited resources they possess, host resources such as CPU, memory, and I/O bandwidth are usually shared among many users and are frequently grossly overloaded, unrepresentative of typical deployment scenarios. Second, in today's overlay testbeds, users cannot perform many privileged operations, including choosing the OS, controlling network stack parameters, or modifying the kernel. Finally, overlay testbeds present a challenging environment for development, debugging, and evaluation [1, 19], three activities which represent a large portion of the work in networked systems research.

To combine the strengths of emulation testbeds (a rich, friendly, controllable environment) with the real network characteristics seen by live-network testbeds, Flexlab replicates these network characteristics inside of an emulator. Ideally, we would create a good *general-purpose* model of the Internet, and use that to drive the emulation, but this is an approach fraught with difficulties. A key obstacle is that a general-purpose emulator, in theory, has a stringent criterion for modeling accuracy: it must yield accurate results for *any* measurement of *any* workload. While much progress has been made on measuring and modeling aspects of the Internet for certain uses, such as improving overlay routing or particular applications [11, 12], creating good general-purpose models of the Internet is still an open problem [6, 7, 10]. Spring *et al.* [20] have made the argument that "reverse-engineering" the entire Internet over a 24-hour period is feasible. The limitation is that it would require an enormous community effort.

Given the difficulty of general-purpose modeling, we focus instead on the simpler problem of modeling the Internet as seen through the lens of an application. Flexlab does this by modeling end-to-end characteristics of Internet paths between pairs of overlay nodes. This reduces both the scale (hundreds of overlay sites vs. millions of Internet nodes) and the complexity (end-to-end measurement vs. detailed routing and queuing models) of the problem domain, making it more tractable.

The two simple models we present use measurements taken by general purpose measurement tools; the first sets static network parameters, and the second updates them dynamically. Our third model, application-centric Internet modeling, takes a much higher-fidelity approach to network conditions: it measures the Internet using traffic similar to that generated by the application under test. This has the advantage that other traffic on the live network reacts similarly as it would to the application itself. It also removes any artifacts that might be introduced by special measurement traffic, and rare or transient network effects are immediately visible to the application. Finally, it is our belief that experimenters will be substantially more trusting of this concrete approach to modeling than to more abstract models.

These models are by no means an exhaustive set, but represent interesting points in the space of Internet models. Many other models are possible. For example, recent work [8, 11, 13] provides novel ways to trade off accuracy for decreased measurement traffic. In addition, models can be made *replayable*, using the same network parameters for multiple runs of an experiment. This enables repeatable experimentation, a feature not possible on the Internet.

Related Work. Network measurement to understand and model network behavior is a popular research area. There has been an enormous amount of work on measuring and modeling Internet characteristics including bottleneck-link capacity, available bandwidth, packet delay and loss, topology, and network anomalies; we cite only a few examples [5, 18, 17, 26]. In addition to use in evaluating protocols and applications, network measurements and models have been used for maintaining overlays [2].

The past few years have also seen growth of experimental network testbeds. Emulab and PlanetLab are the most prominent. The wide-area scope and realism of PlanetLab has attracted several measurement studies that are specific to it [19, 9, 27, 14]. Our work differs from these in its novel bridging of live-network experimentation and emulation.

2 PlanetLab Network & Host Conditions

In this section, we motivate Flexlab in two ways.

Scheduling Accuracy. Today's largest and most public overlay testbed, PlanetLab, is heavily overloaded. It does not represent a realistic deployment environment for many applications, so can cause significant accuracy problems. For example, our measurement agent detailed in Section 5 ran fine on unloaded nodes, but required significant optimization work to produce accurate results on PlanetLab. One of the ways this overload manifests itself is in scheduling jitter. To confirm this, we implemented a test program which schedules a sleep event and measured the actual sleep time using the system clock; the difference between the target and observed wakeup times indicates CPU starvation.

Our scheduling experiment ran on three PlanetLab nodes with differing load averages (roughly 6, 15, and 27) and an unloaded Emulab node running the same OS and kernel; the kernel schedulers run at 1000 Hz. 250,000 sleep events were continuously performed on each node with a target latency of 8 ms, for a total of about 40 minutes. Figure 1 shows the resulting CDF of the additional delay for the sleep events, up to the 90^{th} percentile.

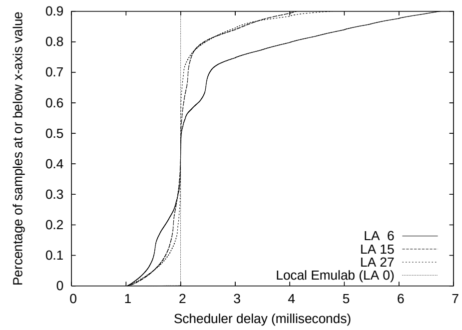

Figure 1: 90^{th} percentile scheduling time difference CDF

On the unloaded Emulab node, the CDF is reduced to a vertical line, due to equal sleep times. On the PlanetLab nodes, 90% of the sleep times are within 2–5 scheduler quanta (milliseconds) of the target, but there is a tail extending to several hundred milliseconds (99.99% of operations return in less than 150 ms). This tail is significant, and poses fidelity problems for programs that are time-sensitive. Many programs will still be able to obtain accurate results, but it is difficult to determine in advance which applications are sensitive to scheduling latency. The effect of scheduling jitter can also be reduced if, as Spring *et al.* [19] point out, tools can be designed so that untimely measurements are discarded. However, this is a difficult proposition for many programs.

Network Conditions and Stationarity. We find that the network conditions seen by PlanetLab can change frequently at small time scales. Importantly, we find the most variability on commodity Internet links, which, while being a minority in PlanetLab, comprise a majority of links in the Internet. To quantify this variability, we ran an experiment that collected high-frequency data on network latency. We do not claim that our experiment captures all interesting variation on these paths; the lesson to be learned from this experiment is that coarse measurement is not sufficient to capture all of the interesting detail of an Internet path.

Our experiment sent pings between pairs of PlanetLab nodes every 2 seconds for a 30 minute period, and analyzed the latency distribution to find "change points" [21]. Change points are points in time when the magnitude of the samples significantly changes; this statistical technique was used by a classic paper on Internet stationarity [28]. We use a method similar to their "CP/Bootstrap" test.

Table 1 shows some of the results. We used representative nodes in Asia, Europe, and North America. One set of North American nodes are on the commercial Internet, and the other are on Internet2. The third column shows the number of change points observed using all gathered data. The fourth column gives the magnitude of the median change for the path as a percentage of the median latency. The second

Path	20 sec. Period Count	2 sec. Period Count	Size %
Asia to Asia	1	2	0.1
Asia to Commercial	0	2	2.9
Asia to Europe	0	4	0.5
Asia to I2	0	6	0.6
Commercial to Commercial	2	20	39.0
Commercial to Europe	0	4	3.4
Commercial to I2	1	13	15.0
I2 to I2	0	4	0.02
I2 to Europe	0	0	—
Europe to Europe	1	9	12.0

Table 1: Change point analysis for latency.

column is derived from the same data, but downsampled to 20-second intervals, simulating lower-frequency measurement. These results show that there are a number of paths that show significant variation at small time scales, and that low-frequency measurement misses nearly all of the change points in such data. The apparent discrepancy between our results and earlier studies, which found much less variation, is explained by our much higher measurement frequency.

3 Overall System Architecture

Flexlab's architecture is outlined in Figure 2. Because we concentrate on emulating end-to-end path characteristics rather than the full Internet topology, Emulab nodes are connected in a full mesh, abstracting the Internet as a set of pairwise network characteristics. An experiment can contain a mix of links modeled by Flexlab and traditional emulated links; each node to participate in a Flexlab link is associated with a PlanetLab node from which it will get its network characteristics. The experimenter can select specific PlanetLab nodes or allow Emulab to select node for them, based on the hosting site or Internet connectivity class (such as commodity Internet, Internet2, or non-North American).

The application under test runs on Emulab hosts and its network operations, such as `connect()` and `send()`, are recorded by the application monitor. The network model, which is easily "pluggable," feeds network parameters into the path emulator, a version of Dummynet [16] that we have enhanced with support for new path characteristics. The network model can also optionally use data from the measurement repository, which currently contains over three million low-frequency path measurements collected from PlanetLab over a period of several months. The model sends network parameters using Emulab's event system, which is a publish/subscribe system. Any node inside or outside of the experiment can publish new characteristics for paths; this makes it easy to implement either centralized or distributed model computation.

Most parts of this infrastructure are user-replaceable, allowing for a wide variety of models. We present three such models in the remainder of this paper; they are intended as beginning points in the exploration of pluggable network models rather than destinations. Our framework enables

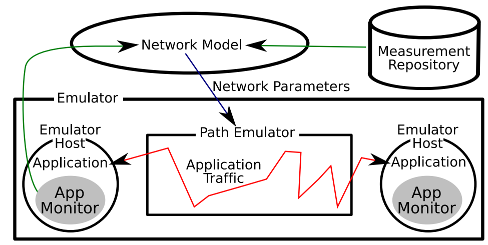

Figure 2: Flexlab Architecture.

transparent switching between different models, and even between Flexlab experiments and running live on PlanetLab. This eases the task of figuring out which network model is appropriate for an application, and enables development/debugging under simple, predictable models with evaluation done under more complex, realistic ones. This system is operational, and we have run a number of experiments using real applications on it.

We base our work on the Emulab network testbed management software, which provides important functionality. Emulab experiments may be interactive or completely scripted, and Emulab provides a distributed event system through which both the testbed software and users can control and monitor experiments. Emulab also provides efficient mechanisms for distributing experimenters' software to nodes, automatic packet trace collection, and gathering of logfiles and other results. Its "PlanetLab portal" [24] extends all of these benefits to PlanetLab nodes, allowing experimenters to easily move back and forth between emulation and live experimentation.

4 Simple Static & Dynamic Network Models

Our first two network models use data collected by our background monitor, which runs constantly on PlanetLab, taking measurements between all site pairs. Because there are a large number of pairs, this background monitoring can only be done at low frequency. We measure latency with `fping` every few tens of minutes, and assess bandwidth using `iperf` every few hours. (We found that less intrusive techniques such as packet-pair and packet-train were too imprecise and inaccurate on PlanetLab.) In the future we will reduce our need for active measurements by doing opportunistic passive measurements of file transfers by CoDeen, a popular CDN deployed on PlanetLab.

This low-frequency data is suitable for determining simple, static network parameters such as average latency and bandwidth. It is archived in the public Datapository [3] with which we federate, where it is available to any researcher. It can also be used by experimenters to choose paths that exhibit some desired characteristic, such as high variability or predictable diurnal variation. Our first network model, the "Simple-Static" model, uses this background information to set network parameters at the beginning of the experiment from historical data, and does not change them thereafter.

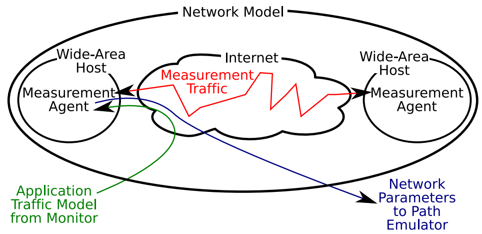

Figure 3: The data flow inside the application-centric Internet model.

As we saw in Section 2, real network conditions are dynamic, so this low-frequency data will not provide sufficient fidelity for many applications. For our next model, "Simple-Dynamic," Flexlab allows the experimenter or application to control measurement frequency, so that paths of interest can be monitored at higher frequency. As those new measurements are obtained, Flexlab continuously adjusts the path emulator parameters. This model uses information from the application monitor, which monitors which nodes actually connect to each other, so we can limit high-frequency measurements to these pairs.

In the future, an important use of the measurements archived in the Datapository will be replay: running emulations with network characteristics recorded in a previous run. With replay, researchers could repeat experiments with some variation of their application or its inputs. However, in such a replay, some applications may select different paths than they did during the original run, meaning that we may have only low-frequency data for those new paths. We can detect this situation, and give the experimenter feedback about the replay's fidelity.

5 Application-Centric Internet Modeling

Our desire to reproduce, with high fidelity, the network conditions that would be seen by an application run on PlanetLab, leads us to a technique we call application-centric Internet modeling. As discussed in Section 1, we do not attempt to trace, reverse-engineer, or model the full Internet. The key insight is that to produce a faithful model, we *need only model the Internet as perceived by the application*—as viewed through its limited lens.

The design of the application-centric Internet model is shown in Figure 3. Referring back to the Flexlab architecture in Figure 2, this is an instance of the "network model" component. The model receives characteristics of the application's offered load from the application monitor, replicates that load on PlanetLab with the measurement agent, determines path characteristics through analysis of the TCP stream, and sends the results back into the path emulator as traffic shaping parameters.

This design has several strong points. First, it creates a feedback loop in which we are constantly adjusting offered loads and emulator settings in near real-time. There is control latency in the communication between the emulator hosts and the wide-area nodes, but this merely time-shifts changes in offered load or network parameters by, typically, tens of milliseconds. Second, the design automatically obtains accurate information on how the network reacts to the offered load. Third, it lets us obtain fine-grained measurements of the traffic we send on PlanetLab, which allows us to track high-frequency network changes, such as we found in Section 2. Finally, it automatically and quickly detects the end-to-end effects of rare events such as outages and route flapping, which can be especially difficult to model.

We make some commonly-made assumptions about the Internet. We assume that most paths have a single bottleneck link, and that the location of that link does not change rapidly (though its characteristics may). We assume that ACK packets are not commonly dropped; missing ACKs are more likely due to forward path congestion. Finally, our work so far focuses only on TCP flows; we plan to extend it to UDP in the future.

5.1 Application Monitor and Measurement Agent

We pair each node in the emulated network with a peer in the live network. The application monitor, introduced in Section 3, runs on each Emulab node and connects to the measurement agent on the corresponding PlanetLab node. In turn, the agent sends updated network parameters to the appropriate path emulator in Emulab.

Application Monitor on Emulab. The applications under test are run with an `LD_PRELOAD` library which informs the monitor process of the application's network calls. Thus, any dynamically-linked executable can be instrumented without modification. The application monitor derives a model of the application's offered network load and sends this model to the measurement agent on the corresponding PlanetLab node. This model is simple and lightweight, consisting of the times and sizes of all `send()`s done by the application. The measurement agent can replicate the application's offered load by performing similarly sized and spaced `send()`s, albeit with different packet contents. By monitoring the application's offered load, rather than the packets it successfully sends on the wire, we see the data rate the application is *attempting* to achieve, rather than what it has been limited to by present network conditions. In some cases, this rate is artificially limited when the socket buffer is full. However, we still capture the true rate while it fills. The monitor also reports on important TCP settings, such as socket buffer sizes.

Measurement Agent on PlanetLab. Whenever the application running on an Emulab node connects to another node inside Emulab, the corresponding measurement agent on PlanetLab likewise connects to the agent on PlanetLab that represents the peer. The agent uses the load model sent by the monitor to generate similar network load, using `send()` calls, while also inspecting the resulting packet stream with `libpcap`. It collects fine-grained information on the TCP connection: for every ACK it receives from the remote agent, it calculates instantaneous throughput and

RTT. From these values it periodically generates and sends parameters to the path emulator in Emulab. To minimize PlanetLab host artifacts, the measurement agent requires little CPU time, and can distinguish between throughput changes due to available bandwidth and those caused by other effects, such as scheduling jitter.

5.2 Path Emulation

We emulate the behavior of the bottleneck router's queue within our path emulator, an enhanced version of the popular Dummynet [16] traffic shaper. The emulation uses two queues: a bandwidth queue, which emulates queuing delay, and a delay queue, which models all other sources of delay: propagation, processing, and transmission. Thus, there are three important parameters: the size of the bandwidth queue, the rate at which it drains, and the time spent in the delay queue. We assume that most packet loss in the wired Internet is caused by congestion, and thus induce loss only by limiting the size of the bandwidth queue.

Since the techniques in this section require that there be application traffic to measure, we bootstrap the model using historical data as in the Simple-Static model. These initial conditions will only be seen by the first few packets; after that, we have higher-quality measurements.

Bandwidth Queue Size and Packet Loss. When the bandwidth queue is full, arriving packets are dropped. The actual bottleneck router in the Internet has a queue whose maximum capacity is measured in terms of bytes and/or packets, but it is difficult to directly measure either of these capacities. Instead, we use a simpler approach: we approximate the size of the queue in terms of time. Sommers *et al.* [17] have proposed using the maximum one way delay to approximate the size of the bottleneck queue. This approach is problematic on PlanetLab because of the difficulty of synchronizing clocks, required to calculate one way delay. However, if we make the assumption that queuing delay along the reverse path does not fluctuate quickly, we can approximate the maximum queuing delay by subtracting the minimum RTT from the maximum RTT. We refine this number by finding the maximum queuing delay just before a loss event, yielding loss episodes consisting of both packets with high RTT and those that have been dropped.

Available Bandwidth. Measuring available bandwidth, the rate at which a flow's packets are drained from the bottleneck queue, has practical subtleties. Some measurement techniques do not take into account the reactivity of other flows in the network. For example, TCP's fairness (the fraction of the capacity each flow receives) is affected by differences in the RTTs of flows sharing the link, but measuring the RTTs of flows we cannot directly observe is difficult or impossible. We avoid these problems by directly measuring the bandwidth available to a specific connection, by sending that flow out into the network and measuring the resulting goodput, averaging it over the last half second to smooth outliers.

Deciding when to change the available bandwidth parameter in the path emulator has subtleties as well. If the application's offered load is not high enough to fully utilize the available bandwidth, we should not cap the bandwidth on the path emulator to this artificially low rate. Thus, we only *lower* the bandwidth available on the emulated path if we detect that we are fully loading the PlanetLab path. If we see a goodput that is higher than the goodput when we last fully utilized the path, then the available bandwidth must have increased, and we *raise* the emulator bandwidth.

Queuing theory shows that when a buffered link is overutilized, the time each packet spends in the queue, and thus the observed RTT, increases for each packet. Alternatively, we note that `send()` calls for a stream tend to block when the application is sending at a rate sufficient to saturate the bottleneck link. In practice, since both of these signals are noisy, we use a combination of them to determine when the bottleneck link is saturated. To determine whether RTT is increasing or decreasing, we find the slope of RTT vs. sample number using least squares linear regression.

Other Delay. The measurement agent takes fine-grained latency measurements. It records the time each packet is sent, and when it receives an ACK for that packet, calculates the RTT seen by the most recently acknowledged packet. We calculate the "Base RTT" the same way as TCP Vegas [4]; that is, the minimum RTT seen recently. This minimum delay accounts for the propagation, processing, and transmission delays along the path, with minimum influence from queuing delay. We set the delay for the delay queue to the Base RTT to avoid double-counting queuing latency, which is modeled in the bandwidth queue.

Outages and Rare Events. There are many sources of outages and other anomalies in network characteristics. These include routing anomalies, link failures, and router failures. Work such as PlanetSeer [27] and numerous BGP studies seeks to explain the causes of these anomalies. Our application-centric model has an easier task: to faithfully reproduce these rare events, rather than find the underlying cause. It automatically observes and mimics features of these rare events that are *relevant* to the application. Outages can affect Flexlab's control plane, however, by cutting off Emulab from one or more PlanetLab nodes. We plan to mitigate that by using an overlay network such as RON.

5.3 Sample Results

Figure 4 shows the throughput of a two minute run of `iperf`, which sends data as fast as possible over a TCP connection. The top graph shows throughput achieved by the measurement agent, which replicated `iperf`'s offered load on the Internet between AT&T and the University of Texas at Arlington. The bottom graph shows the throughput of `iperf` itself, running on an emulated path inside Emulab.

To induce a change in available bandwidth, we sent cross-traffic, in the form of 10 `iperf` steams, on the Internet path between time 35 and time 95. As we can see,

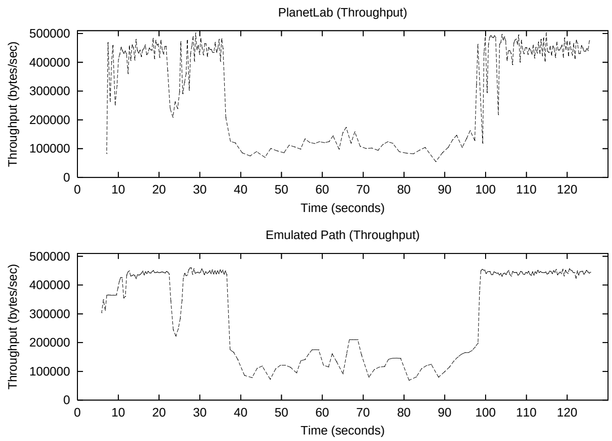

Figure 4: Application-centric Internet modeling, comparing throughput on PlanetLab (top) with the throughput of the application running in Emulab and interacting with the model (bottom).

Flexlab reacts quickly to the change, bringing the throughput of the path emulator down to the new level of available bandwidth. We next point out two other phenomena in this experiment. First, throughput drops in both streams around time 20; that change was presumably caused by cross-traffic from some external source. Second, brief but large drops in throughput occasionally occur in the Internet graph, such as those around time 100. These are due to the measurement agent not getting scheduled for extended periods and thus failing to saturate the link, demonstrating the artifacts due to scheduling jitter discussed in Section 2. The measurement agent correctly determines that these reductions in throughput are not due to available bandwidth changes, and deliberately avoids mirroring these PlanetLab host artifacts on the emulated path.

Acknowledgements

We are grateful to our co-workers for great help with implementation, evaluation, operations, design, and discussion: Kevin Atkinson, Russ Fish, Sachin Goyal, Mike Hibler, David Johnson, Tim Stack, Leigh Stoller, and Kirk Webb; to Dave Andersen and Nick Feamster for the Datapository, to Dave for helpful discussion, to Ken Yocom and the reviewers for their useful comments, and to NSF for its support under grants CNS–0335296, CNS–0205702, and CNS–0338785.

References

[1] J. Albrecht, C. Tuttle, A. C. Snoeren, and A. Vahdat. PlanetLab Application Management Using Plush. *ACM SIGOPS Operating Systems Review*, 40(1):33–40, Jan. 2006.

[2] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *18th SOSP*, pages 131–145, Mar. 2001.

[3] D. G. Andersen and N. Feamster. Challenges and Opportunities in Internet Data Mining. Technical Report CMU–PDL–06–102, Carnegie Mellon University Parallel Data Laboratory, Jan. 2006. http://www.datapository.net/.

[4] L. Brakmo, S. O'Malley, , and L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *SIGCOMM 1994*, pages 24–35, Aug. 1994.

[5] M. Coates, A. O. Hero III, R. Nowak, and B. Yu. Internet Tomography. *IEEE Signal Processing Magazine*, 19(3):47–65, May 2002.

[6] S. Floyd and E. Kohler. Internet Research Needs Better Models. In *First Workshop on Hot Topics in Networks*, Oct. 2002.

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

[8] P. Francis, S. Jamin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A Global Internet Host Distance Estimation Service. *IEEE/ACM Transactions on Networking*, 9(5):525–540, Oct. 2001.

[9] S.-J. Lee et al. Measuring Bandwidth Between PlanetLab Nodes. In *First Passive and Active Measurement Workshop*, Mar. 2005.

[10] X. Liu and A. Chien. Realistic Large-Scale Online Network Simulation. In *2004 ACM/IEEE Conference on Supercomputing*, Nov. 2004.

[11] H. V. Madhyastha et al. iPlane: An Information Plane for Distributed Services. In *Seventh OSDI*, Nov. 2006.

[12] A. Nakao, L. Peterson, and A. Bavier. A Routing Underlay for Overlay Networks. In *SIGCOMM 2003*, pages 11–18, Aug. 2003.

[13] T. S. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *IEEE/ACM Transcactions on Networking*, Oct. 2001.

[14] D. Oppenheimer, B. Chun, D. Patterson, A. C. Snoeren, and A. Vahdat. Service Placement in a Shared Wide-Area Platform. In *2006 USENIX Annual Technical Conf.*, pages 273–288, May–June 2006.

[15] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *First Workshop on Hot Topics in Networks*, Oct. 2002.

[16] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM CCR*, 27(1):31–41, Jan. 1997.

[17] J. Sommers, P. Barford, N. Duffield, and A. Ron. Improving Accuracy in End-to-end Packet Loss Measurement. In *SIGCOMM 2005*, pages 157–168, Aug. 2005.

[18] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *SIGCOMM 2002*, pages 133–145, Aug. 2002.

[19] N. Spring, L. Peterson, V. Pai, and A. Bavier. Using PlanetLab for Network Research: Myths, Realities, and Best Practices. *ACM SIGOPS Operating Systems Review*, 40(1):17–24, Jan. 2006.

[20] N. Spring, D. Wetherall, and T. Anderson. Reverse-engineering the Internet. In *Second Workshop on Hot Topics in Networks*, Nov. 2003.

[21] W. A. Taylor. Change-Point Analysis: A Powerful New Tool for Detecting Changes. http://www.variation.com/cpa/tech/changepoint.-html, Feb. 2000.

[22] A. Vahdat et al. Scalability and Accuracy in a Large-Scale Network Emulator. In *Fifth OSDI*, pages 271–284, Dec. 2002.

[23] A. Vahdat, L. Peterson, and T. Anderson. Public statements at PlanetLab workshops, 2004–2005.

[24] K. Webb et al. Implementing the Emulab-PlanetLab Portal: Experience and Lessons Learned. In *First Workshop on Real, Large Distributed Systems*. USENIX Association, Dec. 2004.

[25] B. White et al. An Integrated Experimental Environment for Distributed Systems and Networks. In *Fifth OSDI*, pages 255–270, Dec. 2002.

[26] K. Xu, Z.-L. Zhang, and S. Bhattacharyya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *SIGCOMM 2005*, pages 169–180, Aug. 2005.

[27] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services. In *Sixth OSDI*, pages 167–182, Dec. 2004.

[28] Y. Zhang, N. Du, V. Paxson, and S. Shenker. On the Constancy of Internet Path Properties. In *SIGCOMM Internet Measurement Workshop*, pages 197–211, Nov. 2001.