A Platform for Unobtrusive Measurements on PlanetLab

Rob Sherwood Neil Spring
{capveg,nspring}@cs.umd.edu

Abstract

TCP Sidecar is a network measurement platform for injecting probes transparently into externally generated TCP streams. By coupling measurement probes with non-measurement traffic, we are able to obtain measurements behind NATs and firewalls without alerting intrusion detection systems. In this paper, we discuss Sidecar's design and our deployment experience on PlanetLab. We present preliminary results from Sidecar-based tools for RTT estimation ("sideping") and receiver-side bottleneck location ("artrat").

1 Introduction

Internet measurement is key to optimizing performance, building overlay topologies, developing improved transport protocols, understanding the influence of network policy, and many other research tasks [4]. Yet the scope and detail of network measurement is limited more by the potential for soliciting abuse reports and administrative headache than by the bandwidth required to measure every interesting property [20]. Traffic designed to measure the network is often out-of-the-ordinary, interpreted by intrusion detection systems (IDSs) as anomalous or as attempts to exploit unknown vulnerabilities. To make a network measurement "safe," not just for the network but also to avoid abuse reports, requires techniques beyond those of Scriptroute [21]: it requires a fundamental shift in the design of network measurement probes and responses.

We present a measurement platform for reduced intrusiveness called TCP Sidecar. Sidecar's main insight is that soliciting abuse reports and triggering IDSs can be avoided by injecting carefully-crafted probes into externally-generated, non-intrusive network traffic. Where typical measurement tools select which hosts to probe and in what order, Sidecar does not control the source, destination, or the exact time of the measurement. Much like a sidecar attaches to a motorcycle, TCP Sidecar attaches to a TCP connection and is just "along for the ride." The Sidecar is also a container: it can carry various measurement techniques for discovering different network properties.

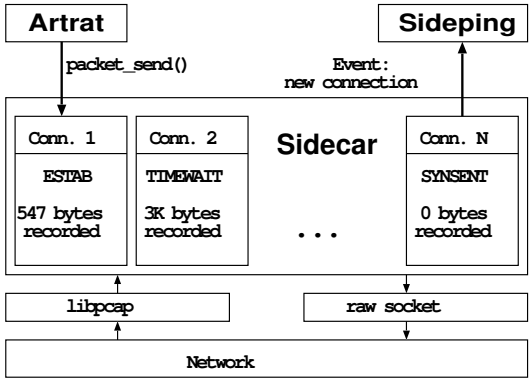

Figure 1: Sidecar is a platform for unobtrusive measurements that provides an event-driven interface and connection tracking to higher-level tools, e.g., artrat, sideping.

In this paper, we describe Sidecar-based topology inference, round-trip time measurement, and bottleneck location. We show how Sidecar obtains measurements, through network address translators (NATs) and firewalls, unavailable to traditional measurement techniques. Also, we describe our experience with Sidecar on PlanetLab. In a complementary paper, we describe Passenger [17], a Sidecar-based tool that makes use of the IP record route option for topology discovery.

This paper is organized as follows. In Section 2, we describe the Sidecar platform and API. We present our experience from running Sidecar on PlanetLab in Section 3 and two examples of Sidecar-based tools in Section 4. Last, we describe our plans for future work in Section 5.

2 Sidecar Design

Sidecar (Figure 1) is a platform that supports transparently injecting measurement into TCP streams. Probes consist of acknowledgments and replayed data segments, carefully crafted not to interfere with the ongoing TCP connection. Sidecar requires no modification to end-points, requires no firewall rules (unlike Sting [16]), and can run at either end-point of a stream or even in a network middle box. Sidecar's only requirement is that it be on both the forward and reverse paths of a connection. Sidecar probes require an external source of TCP traffic, but the

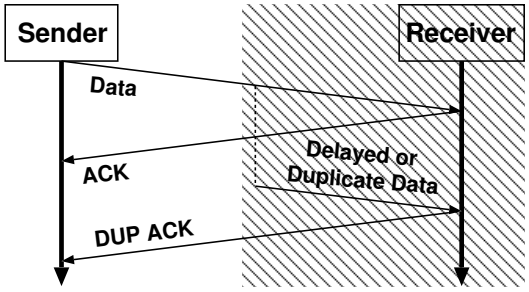

Figure 2: Sender incorrectly assumes (shaded region) that duplicate ACKs are from delayed, reordered, or duplicated packets.

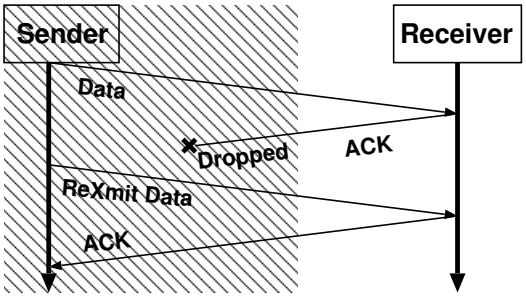

Figure 3: Receiver incorrectly assumes (shaded region) that probes are valid retransmissions from sender due to lost ACK.

characteristics of the application being instrumented matter little.

2.1 Unobtrusive Probing

Sidecar probes are TCP packets that look like retransmitted data. Upon receiving retransmitted data, TCP receivers send a duplicate ACK because the original ACK could have been lost (Figure 3). TCP senders ignore single duplicate ACKs because they could be caused by delays (Figure 2) or reordering in the network. Sidecar records application data passively so that segments can be retransmitted accurately (Figure 4).[1] Because packet loss and duplication are expected in TCP, IDSs are unlikely to generate alerts from Sidecar probes. Thus, Sidecar probes solicit responses from end-hosts without affecting applications or alerting IDSs.

Because Sidecar probes seamlessly attach and follow application streams, they can reach places unsolicited probes cannot. For example, if a Sidecar-enabled tool instrumented web server traffic, Sidecar probes could follow web connections from the server back to the corresponding web clients, even if they were behind firewalls or NATs.

The size of the probe can be varied by changing the amount of traffic replayed, only limited by the connection MTU and the amount of data recorded. Probes can be sent even after the connection has closed by replaying the final FIN-ACK packet, as long as the receiver is in the TIME-WAIT state. The last is possible because the final ACK of the three-way close might have been lost, so replaying the FIN-ACK causes a retransmission of the final ACK.

Typically, a Sidecar-enabled tool would further modify probes. For example, one could implement a Sidecar traceroute-like [8] topology discovery tool by setting the IP TTL field of the Sidecar probe to 1, and then incrementing until an ACK was received from the end-host. With Sidecar running on a web server, this tool would

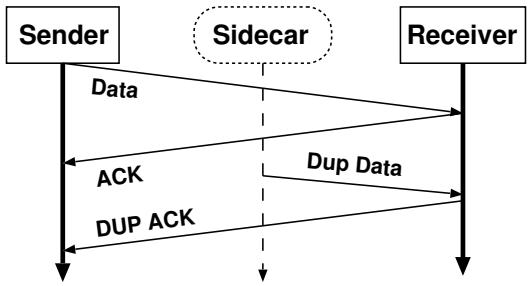

Figure 4: Reality: Sidecar probes are replayed data packet that generate duplicate ACKs. Probes are transparent to both sender and receiver applications.

obtain the path back to any client without out-of-stream packets.[2]

TTL-limited Sidecar probes can also detect NATs. If a probe is sent to IP address A at TTL=t, but the response is an ICMP time-exceeded message with source address A, we can infer that there is a NAT at hop t. We can then continue to increase the TTL to find the actual distance to the end-host, effectively probing behind the NAT. Passenger [17] is a Sidecar-enabled topology discovery tool that combines TTL-limited traceroute data with data from the IP record route option. We present two further examples of Sidecar tools in the Section 4.

2.2 Sidecar API

The Sidecar API (Figure 1) provides connection tracking, probe identification, round trip time estimation as well as bandwidth and memory usage limits. The Sidecar tools are event-driven applications that receive event notifications such as new connections, incoming and outgoing probes. The Sidecar initialization function takes a libpcap [9] filter string, e.g., "host www.google.com and port 80", as a parameter, and ignores events that do not match the filter. To construct packets for retransmissions, Sidecar tracks state for each connection, including sequence

[1] Paxson [12] notes that retransmitted data can change the data stream sent if the original and retransmitted data are not consistent.

[2] Discovering the topology between server and web clients is precisely the measurement by Padmanabhan *et al.* [11].

numbers and the last 3000 bytes (two full standard Ethernet packets) of application data in both directions. Sidecar automatically matches probes to their sent and received libpcap timestamps for increased accuracy over gettimeofday() [19]. Sidecar differentiates probes from legitimate traffic by changing the probe's IP identifier field.

3 Sidecar on PlanetLab

In this section, we discuss the lessons learned by running Sidecar on PlanetLab. We divide these lessons into two categories: problems we expected that turned out to be non-issues, and problems we did not expect.

3.1 Non-Issues

No abuse complaints from embedded probes. We run a Sidecar-based topology discovery tool, Passenger [17], for seven days on all traffic generated by the CoDeeN [22] web-proxy project. CoDeeN is a content distribution network hosted on PlanetLab that supports approximately 1 millions requests per day [3]. Of the 13,447,011 unique IP addresses, we ran a Sidecar based traceroute scan back to each client, using the algorithm described in Section 2. No abuse reports were generated from our probes.

PlanetLab VNET worked with Sidecar. PlanetLab implements a connection tracking and traffic isolation system called VNET [7] to prevent researchers from interfering with each other. With VNET, all connections are owned by a specific slice, and slices can only read and write raw packets that come from connections that they own. It was not immediately clear that Sidecar would be compatible with VNET, because Sidecar assumes that processes in the same slice can write to each other's connections and slices can write packets to sockets after they have gone to the timewait state. It is a measure of the success of the VNET design that it was able to accommodate Sidecar.

3.2 Unanticipated Issues

Clocks changed and went back-in-time. PlanetLab machines run on a variety of hardware and loads, causing variable clocks and inconsistent measurements. As part of our future work, we are adding a periodic sanity check to Sidecar to compare the elapsed time to the elapsed processor cycles as returned by the *RDTSC* instruction. In this way, Sidecar can notify a Sidecar tool that significant clock skew has occurred, and to adapt accordingly, potentially discarding timing data.

Libpcap on PlanetLab drops and reorders packets. Packets drops and reordering occur more frequently on PlanetLab than our development machines. Particularly problematic was that the final ACK of the three-way-handshake would appear before the SYN-ACK packet. As a result, Sidecar's connection tracking had to be rewritten to be more resilient to these issues.

Firewalls unset DF. In an attempt to reduce the number of packets traversing libpcap, we decided to mark probe packets for which the sent timestamp was unimportant (those that are merely payload intended to cause delay, as in RPT [6]) to separate them from important traffic in the libpcap filter. We marked uninteresting packets by unsetting the Don't Fragment (DF) bit in the IP header, and adjusted our libpcap filter appropriately However, firewalls around some PlanetLab nodes unset the DF bit on incoming packets, foiling our scheme.

IO system calls occasionally took seconds. We saw intermittent multi-second delays when running Sidecar. Using *strace -T*, we found that some open() and write() system calls would take seconds to complete. Because the problem was intermittent, we could not isolate the cause.

PlanetLab web servers don't implement persistent connections. RedHat Fedora Core 2, PlanetLab's base distribution, ships with persistent web connections disabled, despite RFC2616's recommendation that they *should* be enabled. Many of the results in Section 4 used connections from one PlanetLab machine to the web server of another PlanetLab machine as the source of external traffic. The lack of persistent connections shortened connection time so the majority of PlanetLab-to-PlanetLab measurements relied on the post-connection FIN-ACK Sidecar probes as described above.

Sidecar required resource limits Because Sidecar probes are triggered in response to external traffic, and the rate of external traffic is not under Sidecar's control, it quickly became necessary to implement resource metering. Sidecar implements an internal rate limiting scheme on all outgoing probes and monitors the size of the outgoing queue. If the queue size exceeds a threshold value, Sidecar ignores new connections until the queue falls below the threshold. In this way, Sidecar tools need not be exposed to the underlying details of the connection tracking, traffic bursts, or rate limiting.

Generate artificial traffic carefully Sidecar is unobtrusive because it attaches to pre-existing traffic sources. However, for testing or probing specific portions of the network, it is sometimes useful to artificially generate a seemingly legitimate traffic source for Sidecar. In one experiment [17], we created a custom web client to visit a list of 160,000 websites from each PlanetLab node and mimic the presumed innocuous behavior of a web crawler. For each web server, the custom web client connected and performed a full HTTP session while Sidecar attached to the traffic stream to send Sidecar probes.

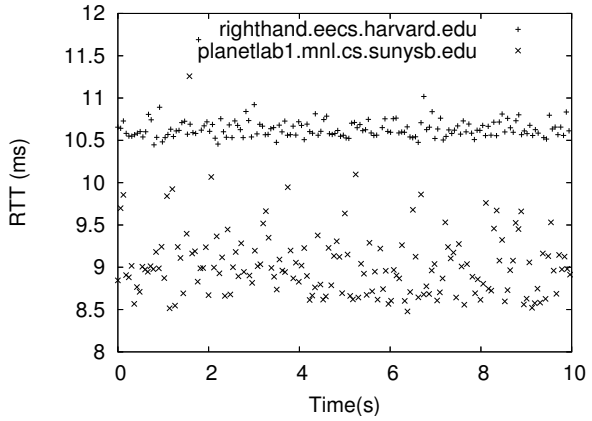

Figure 5: Sideping RTT measurements from UMD to two ICMP echo filtered PlanetLab nodes.

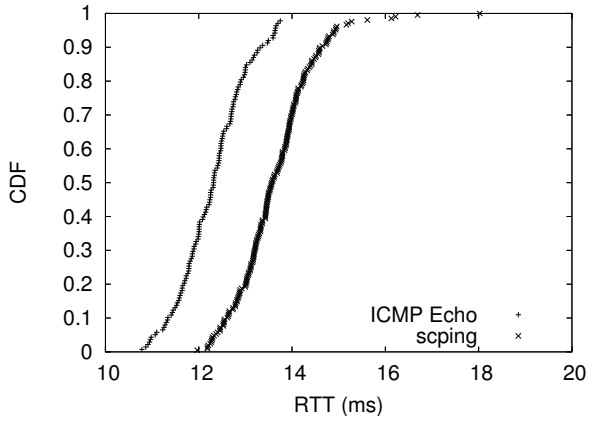

Figure 6: Sideping RTTs vs ICMP Echo: Difference exposes NAT + wireless network.

This experiment generated ten abuse reports, but surprisingly from the web traffic, not the Sidecar probes. The reports were not triggered by automated intrusion detection systems, but apparently by administrators noting the similarity of PlanetLab machine names after manual inspection of HTTP access logs entries. We failed to anticipate the prevalence of virtual hosting and the need to randomize the list of websites. The first caused individual clients to query a single server repeatedly, increasing the number of log entries, and the second caused PlanetLab clients to accidentally synchronize and query the same server simultaneously, decreasing the time between log entries. These issues were exacerbated due to a coding error where the *User-agent* string in the HTTP requests had been reset to a default "Mozilla"-like string. We believe that if the correct User-agent string had been in place, i.e., one pointing to an explanatory web page with contact information, fewer abuse reports would have been directed to PlanetLab.

We plan to explore new less intrusive, artificial traffic generation techniques for future Sidecar experiments.

4 Sidecar Tools

We present two examples of reduced intrusiveness Sidecar-based tools that suggest the generality of the platform. The first tool, sideping, provides accurate round trip measurements with increased accuracy. The second tool, artrat, performs bottleneck location at the receiving end of a connection. As Sidecar modules, both require a separate source of connections, though we use a driver that creates new connections on demand for debugging. In other work, we describe Passenger [17], a Sidecar-enabled tool that combines traceroute probes with the record route IP option for topology discovery.

4.1 sideping: Round Trip Time Estimator

Sideping estimates network latency by measuring the round trip time of Sidecar probes in a TCP connection. Although latency is an unsophisticated measurement, the extensive use of the all-pairs PlanetLab ping [13] data set demonstrates its importance.

Sideping seeks to avoid over- and underestimation of round trip times. Tools like ping can overestimate RTTs because they assume that the probe's sent time is close to the sendmsg() call. By contrast, Sidecar records the timestamp from libpcap for the time the probe was given to the network interface device. Rate limiting means that the probe can reach the interface well after the application asked it to be sent. Ping can also underestimate RTT when probing a host behind a NAT. Because sideping can follow TCP connections to their end-points, researchers can gain insight into network dynamics behind NATs. Compared to TCP's internal RTT estimation protocol, sideping does not inflate RTTs by including delayed ACK time.

Figure 5 shows sideping collecting previously-unavailable RTT measurements from two PlanetLab nodes that filter ICMP echo packets. Figure 6 shows the difference between ICMP echo and sideping RTT measurements traversing a NAT to a wireless network. The ICMP echo reply packets return with a larger TTL than the sideping responses. The difference between the two techniques, 0.797 ms on average, is extra delay from the wireless network.

4.2 artrat: Receiver-side bottleneck detection

Artrat[3] is a Sidecar-based tool that attempts to locate local bottlenecks, from the receiver's perspective. This information could be used to decide whether local network resources were sufficiently-provisioned or if they should be upgraded. Although tools [1, 6] exist to perform bot-

[3] Artrat: Active Receiver TCP Rate Analysis Tool

tleneck location by instrumenting the sender, we believe we are the first to focus on the perspective of the receiver. We believe that this tool will be of use to PlanetLab researchers who are concerned with local bandwidth conditions during their experiments.

Artrat correlates the congestion delay in the connection with the queuing delay at local routers. Sidecar reports any router whose queuing delay correlates over time with the congestion delay as a suspected bottleneck. Similar to TCP Vegas [2], we measure the congestion delay as the difference between the current RTT and the baseline (minimum) RTT.

To measure the queuing delay at routers, artrat first discovers the router, a, five hops[4] into the network using a TTL limited probe. Then, artrat periodically sends ICMP echo probes with the IP timestamp option [14] to router a, and parses the response (Figure 7). The IP timestamp option records the time at each router in milliseconds and (by RFC792) the ICMP echo response packet has the same options payload as the echo request. In this way, artrat learns the local time of each router along the outgoing path to a, and, most importantly, of each router on the path from a back to the receiver. Similar to our definition of congestion delay above, we define the queuing delay between two routers as difference between the current jitter and the minimum observed jitter. If we label the IP option timestamps for the jth probe as $S_{1,j} \ldots S_{9,j}$ and call $S_{0,j}$ and $S_{10,j}$ the send and received times for the ICMP probe j, we can calculate the queuing delay, $q_{i,j}$, between router i and $i+1$ as computed by the jth probe as:

$$q_{i,j} \;=\; S_{i+1,j} - S_{i,j} \;-\; min_k(S_{i+1,k} - S_{i,k})$$

Then we compute the correlation between the congestion delay and $q_{i,j}$ for all routers i, and output the $i \rightarrow i{+}1$ link with the highest correlation as the likely bottleneck.

We ran artrat on a local network testbed to test the scheme. The testbed consisted of a client connected with a 10Mbps Ethernet card to a 100Mbps network. We ran artrat while the network was idle (Figure 8) and while downloading a 20MB file (Figure 9). When the network was idle, artrat found no significant queuing delay. While the network was in use, artrat successfully found queuing delay on the inbound portion of the 10Mbps link (labeled "1→ R" in Figure 9).

The coefficient of correlation between the congestion delay and routing delay at router 1 in Figure 9 is 0.24. Although this is low, the second highest coefficient of correlation was 0.072, so artrat successfully found the 10Mbps link as the bottleneck. This correlation analysis simply compared the ith ICMP probe with the ith Sidecar probe, ignoring timing information and dropped probes. Implementing robust time-series analysis techniques is future work, but the technique shows promise.

[4] *Five* hops into the network was chosen because the IP header limits the number of recorded timestamps to nine. If the local path is symmetric, five hops is the maximum distance the probe can travel away from the receiver so that the return path does not exhaust the IP option array.

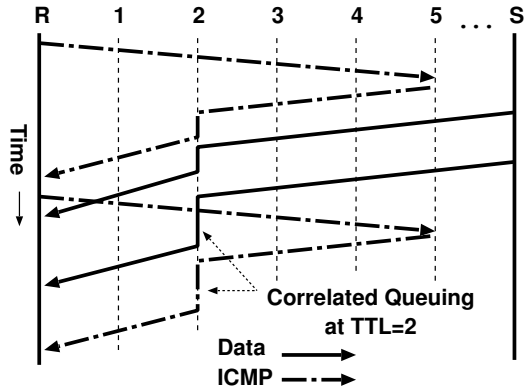

Figure 7: Overview: Artrat correlates congestion and queuing delays to do receiver-side bottleneck location (example: bottleneck from S to R at TTL=2).

Artrat makes two assumptions that must be validated before use. First, due to the baseline measurements, this technique is sensitive to clock skew, both locally and at routers. Thus, artrat must periodically compare the time elapsed on all clocks against some external source, like the *RDTSC* instruction or a remote NTP server, using the techniques of Moon *et al.* [10]. Second, artrat requires some symmetry in local routing. Specifically, artrat can only discover bottlenecks on the path of the returning ICMP probe. It is the subject of future work to integrate artrat with topology-aware tools to verify the symmetric nearby network assumption (perhaps using remote traceroute servers as in Rocketfuel [18] or another service).

While one could create a version of non-Sidecar enabled artrat, the Sidecar version benefits from increased accuracy in RTT measurements (Section 4.1) and already written connection tracking libraries.

5 Conclusion and Future Work

TCP Sidecar provides a platform for non-intrusive network measurements by injecting probes into external sources of traffic. One potential source of external traffic is PlanetLab, which supports many high volume, publicly accessible services, e.g., CoDeeN [22], OpenDHT [15], Meridian [23], and CoralCDN [5]. Many of these services perform their own measurements, and might benefit from less intrusive Sidecar-based approaches. This creates an interesting symbiotic relationship between measurement studies and PlanetLab hosted services.

In our future work, we plan to complete a large-scale Sidecar-based measurement study. Our current work is

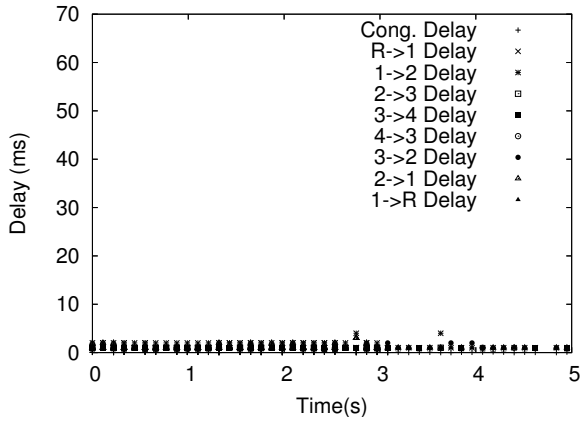

Figure 8: Artrat Experiment: Idle connection: no bottlenecks.

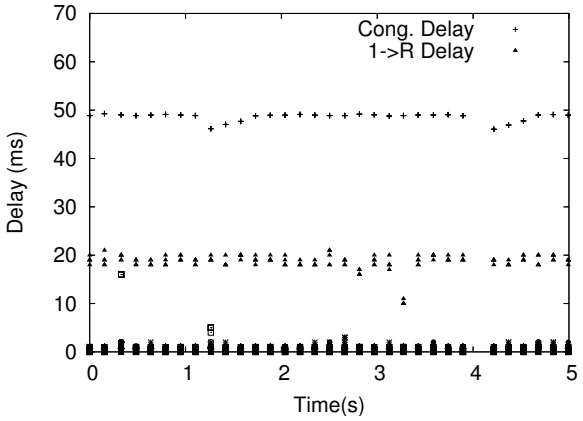

Figure 9: Artrat Experiment: Data Transfer: bottleneck at 1→R, i.e., 10Mbps link. (Data labels as in Figure 8)

in concert with the CoDeeN project, but we plan to expand to other PlanetLab services to avoid host selection bias. We also plan to better document the Sidecar API so that other researchers might benefit from our work. Sidecar is available for download at http://www.cs.umd.edu/projects/sidecar.

Acknowledgments

We owe Mark Huang and the rest of the PlanetLab staff for their help and patience in the early phases of debugging Sidecar and its tools. We would also like to thank Vivek Pai and the rest of the CoDeeN project for allowing us access to their slice, and Fritz McCall and the UMIACS staff for their timely assistance. Last, we would like to thank our shepherd, Dina Katabi, and the anonymous reviewers for their helpful comments. This work was supported by grants ANI 0092806 and CNS-0435065 from the National Science Foundation.

References

[1] A. Akella, S. Seshan, and A. Shaikh. An empirical evaluation of wide-area internet bottlenecks. In *Internet Measurement Conference*, 2003.

[2] L. Brakmo and L. Peterson. TCP Vegas: End to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communication*, 13(8):1465–1480, 1995.

[3] http://codeen.cs.princeton.edu/talks/iris_pl.

[4] Computer Science and Telecommunications Board, National Research Council. *Looking Over the Fence at Networks: A Neighbor's View of Networking Research*. The National Academies Press, 2001.

[5] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with Coral. In *NSDI*, 2004.

[6] N. Hu, *et al.* Locating internet bottlenecks: algorithms, measurements, and implications. In *ACM SIGCOMM*, 2004.

[7] M. Huang. VNET: PlanetLab virtualized network access. http://www.planet-lab.org/doc/vnet.php, 2005.

[8] V. Jacobson. Traceroute. ftp://ftp.ee.lbl.gov/traceroute.tar.Z.

[9] http://www.tcpdump.org.

[10] S. B. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. In *INFOCOM*, 1999.

[11] V. N. Padmanabhan, L. Qiu, and H. J. Wang. Passive network tomography using bayesian inference. In *IMW*, 2002.

[12] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31:2435–2463, 1999.

[13] http://pdos.csail.mit.edu/~strib/pl_app.

[14] J. Postel, editor. Internet protocol specification. IETF RFC-791, 1981.

[15] S. Rhea, *et al.* OpenDHT: A Public DHT Service and Its Uses. In *Proceedings of ACM SIGCOMM 2005*, 2005.

[16] S. Savage. Sting: a TCP-based network measurement tool. In *USITS*, 1999.

[17] R. Sherwood and N. Spring. Touring the Internet in a TCP Sidecar. In *ACM Internet Measurement Conference (IMC)*, 2006.

[18] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring ISP topologies with Rocketfuel. *IEEE/ACM Transactions on Networking*, 12(1):2–16, 2004.

[19] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using PlanetLab for network research: Myths, realities, and best practices. *ACM SIGOPS Operating Systems Review*, 40(1):17–24, 2006.

[20] N. Spring, D. Wetherall, and T. Anderson. Reverse-engineering the Internet. In *HotNets*, 2003.

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

[22] L. Wang, *et al.* Reliability and security in the CoDeeN content distribution network. In *USENIX Annual Technical Conference*, 2004.

[23] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: A Lightweight Network Location Service without Virtual Coordinates. In *ACM SIGCOMM*, 2005.