Scamper: a Scalable and Extensible Packet Prober for Active Measurement of the Internet

Matthew Luckie
Department of Computer Science
University of Waikato
Hamilton, New Zealand
mjl@wand.net.nz

ABSTRACT

Large scale active measurement of the Internet requires appropriate software support. The better tools that we have for executing consistent and systematic measurements, the more confidence we can have in the results. This paper presents scamper, a powerful open-source packet-prober for active measurement of the Internet designed to stand alone from coordination mechanisms. We built scamper and populated it with specific measurement techniques, making design decisions aimed at allowing Internet researchers to focus on scientific experiments rather than building accurate instrumentation.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Measurement

Keywords

Active measurement, Software, Tools

1. INTRODUCTION

Packet probing experiments capture relatively simple measurements – typically delay, loss, reordering, and topology – that yield surprising insights into the structure and behaviour of the Internet. An early measurement experiment investigating the effect of packet size on delay suggested that TCP implementers should refine the algorithm used to compute the TCP RTO value [1]. The massive scale and continuing growth of the Internet has increased the complexity of measuring it, both technically and methodologically. In the past decade, researchers have built and operated many large-scale active Internet measurement platforms [2, 3, 4, 5, 6, 7, 8, 9] where each typically reproduces a tremendous amount of supporting software development. Faced with continuing funding constraints, members of the Internet measurement community organised a workshop in 2005 to strategise a way to collaborate on creating and operating a community-oriented network measurement infrastructure; the workshop report [10] presents many goals and desires, including the need for better ways to organise large-scale measurements.

This paper focuses on a small part of this problem; building a packet-prober that makes it easy to conduct large-scale measurements and archive collected data in a well-defined format. A packet-prober should abstract away problems of coordinating individual measurements on a host, provide APIs that deal with operating system differences, provide APIs for obtaining accurate timing information, and produce output that is more detailed and easier to process than that produced by existing system tools. By building a packet-prober with these goals in mind we can help the experimental researcher avoid the system programming and administration problems of coordinating individual measurements on a single host. Rather than suffering the overhead associated with parallelisation of a single instance of a given measurement process, a researcher can rapidly implement a new measurement technique and focus on the analysis and validation of results.

Such community software infrastructure will also improve the capability of researchers to leverage measurements from volunteers across the global Internet. At present researchers need to be careful when soliciting volunteers to do a measurement that uses the system's included traceroute as the implementation included in each operating system differs remarkably in operation and utility, even though most are based on Van Jacobson's implementation [11]. For instance, none support Paris traceroute [12], FreeBSD's implementation supports TCP probing but does not process TCP responses, and Ubuntu's implementation can trigger rate limiting because it sends bursts of probes.

This paper presents scamper, a powerful packet-prober designed to support large-scale Internet measurement. Included in the contribution to the measurement community are feature-rich implementations of traceroute, ping, MDA traceroute, four alias resolution techniques, Sting, and parts of TBIT. We begin in section 2 by first establishing a set of design objectives. Sections 3, 4, 5, and 6 present a technical overview of scamper, detailing its implementation, features, performance, and use. Section 7 shows how scamper has been useful in conducting research, including both our experiences and those of others. Finally, section 8 outlines related work and section 9 concludes.

2. DESIGN OBJECTIVES

Precise: promote good science by ensuring details found in a response packet are easily available to measurement techniques, easily stored, and then easily read by analysis routines. A researcher should be able to annotate both individual measurements and a collection of measurements describing the purpose and specifics of the data collection. Packet timestamps should be recorded from the best available source – whether it be from user-space or the network interface. Decoding a chain of packet headers can be tedious and error prone; therefore, the packet-prober should present decoded packet headers to a measurement technique so that the implementer can easily use them. Finally, the details found in a packet header should be preserved in a data file even if the need to do so has not yet been established.

To motivate this last point, consider macroscopic Internet topology mapping using traceroute; traditionally, the data recorded per-hop is an IP address, the RTT measured, and the TTL used in the probe packet. However, Augustin *et al.* recently brought to the community's attention the problem of routers that forward traceroute probes with a TTL of zero, which results in a subsequent router appearing twice in a path and therefore a false link inference [12]. Unfortunately, few topology mapping projects record the TTL found in the ICMP quotation, preventing the correction of link inferences made using topology data archived years earlier.

Parallelised: efficiently manage the several hundred concurrent measurements that are a reality of macroscopic Internet measurement. For Internet-scale experiments to complete in a reasonable length of time requires concurrent measurements. Active measurements can spend a lot of time waiting, for example, for a reply to a TTL-limited probe in traceroute, or for a reply to a TCP SYN probe. The period of time where productive work is done is when sending a probe and receiving a reply, so measurement is most productive when we are able to probe at a constant rate. However, an architecture that allocates a thread per measurement does not scale as well as an event-based system. A packet-prober should provide APIs that allow resources such as sockets to be shared amongst measurements. Doing so improves performance by reducing the number of file descriptors that the operating system has to service when multiplexing synchronous I/O.

Portable: support a wide range of operating systems and computer architectures. The requirements of some active measurement techniques are not well catered for by the Berkeley sockets API, which is designed for general-purpose Internet software. The methods available to bypass the sockets API differ amongst operating systems, so a significant volume of code has to be implemented for any one technique to be portable across systems. The experimental researcher would benefit from an API that is operating system agnostic and reduces the code required to create and send a packet and then receive and decode a response.

Flexible: each experiment has different motivations and requirements; a packet-prober should be flexible enough to support them. While some experiments can be specified as a one-shot measurement where the order in which a measurement is conducted is not important, some measurements are conducted in reaction to a previous measurement [13], or require some amount of coordination and control. These requirements call for the packet-prober to be flexible in the ways it allows a researcher to specify measurements.

Volunteer friendly: some measurements are made by volunteers that run experiments on behalf of researchers. A packet-prober should therefore be easily compiled on systems for which a pre-compiled binary is unavailable; easily operated by a volunteer; lightweight enough to run in a resource constrained environment such as an embedded system; and self-contained so that a potential volunteer will not need to install a suite of dependencies.

Modular: support multiple measurement techniques, and be easily extended to include techniques that have not yet been devised.

3. SCAMPER

Guided by the design objectives listed in section 2 we implemented scamper, a parallelised packet-prober capable of large-scale Internet measurement using many different measurement techniques. Figure 1 illustrates the overall architecture of scamper. Briefly, scamper obtains a sequence of measurement tasks from the input sources and probes each in parallel as needed to meet a packets-per-second rate specified on the command line. Tasks currently being probed are held centrally by scamper in a set of queues – the probe queue if the task is ready to probe, the wait queue if it is waiting for time to elapse, and the done queue if the task has completed and is ready to be written out to disk. Each measurement technique is implemented in a separate module that includes the logic for conducting the measurement as well as the input/output routines for reading and writing measurement results, allowing measurement techniques to be implemented independently of each other. When a new measurement task is instantiated, the task attaches a set of callback routines to itself that scamper then uses to direct the measurement as events occur, such as when it is time to probe, when a response is received, or when a time-out elapses. Sockets required as part of a measurement are held centrally by scamper in order to share them amongst tasks where possible so that resource requirements are reduced. Finally, scamper centrally maintains a collection of output files where completed measurements are written.

Output: scamper provides two output file formats; an ASCII text option, and a binary file format known as *warts*. The text option produces low-fidelity output similar to the ping and traceroute utilities and is suitable for interactive use. The binary option is an extensible format designed for use by researchers because of its ability to record detail and provide archival features. Scamper includes a library that allows its binary output files to be easily read, and CAIDA have created a Ruby library [14] which allows researchers to develop analysis programs in Ruby. Scamper supplies an API to assist a researcher implementing a new measurement technique to create a record which can then be stored in the binary file format. To promote precision and discourage researchers from recording results in the text option, the library provides no ability to read results from a text file.

Portability layer: scamper provides a portability layer that shields a researcher implementing a measurement technique from the differences in the APIs of each operating system. An author of a measurement technique uses scamper's portability layer to define the details of a probe packet and to transmit it, and has the corresponding responses decoded and passed to the task by its specified callback routines. The portability layer hides, for example, the details of which byte order the system expects packet headers to be supplied in,

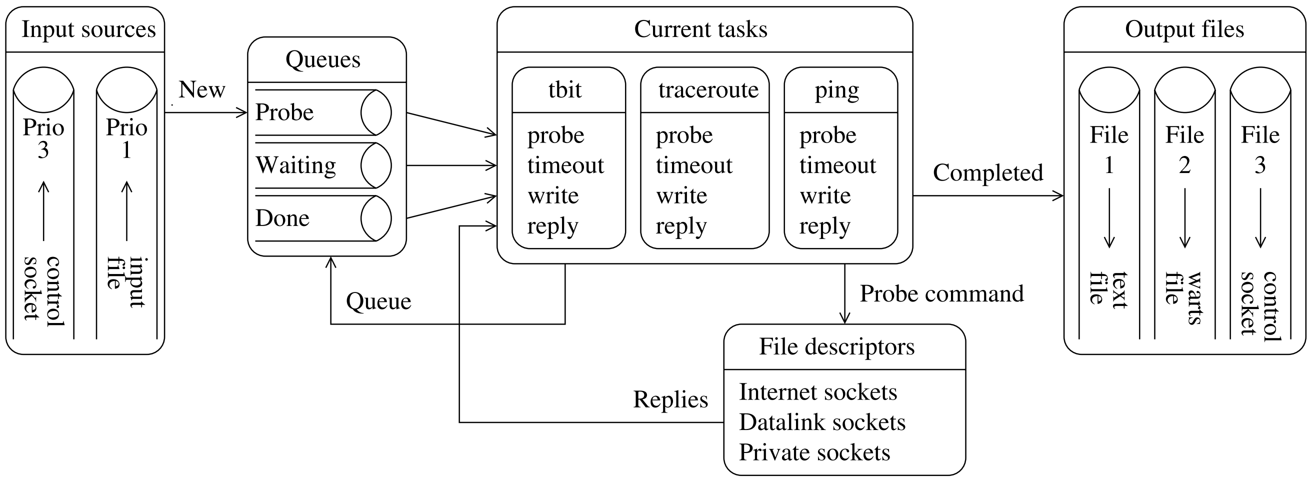

Figure 1: The architecture of scamper. Measurement tasks are supplied from one or more input sources, including from an input file, from the command line, or from a control socket. A scamper task is abstract; the exact behaviour is determined by a set of implementer-provided callback functions. Current tasks can be in one of three states; waiting to probe, waiting for a response or to timeout, or waiting to be written to disk. Completed tasks can be written to a text file or a binary file, or over a control socket for interpretation.

and the byte order that values in packet headers in response packets are returned in. If the system would interfere in transmitting a packet – e.g. by silently fragmenting a probe or re-writing packet header fields, scamper allows a technique to use the datalink sockets instead. Because datalink sockets expect a complete frame including layer-2 headers, scamper provides an API to obtain the layer-2 addresses dynamically using neighbour discovery protocols. Similarly, a technique can use a datalink socket if the system does not provide the ability to read particular types of packets from an Internet socket, such as raw TCP frames. Finally, some techniques that use TCP as their probe method, such as Sting [15] and TBIT [16], require the ability to prevent the operating system's TCP stack from interfering in a measurement by responding to unexpected packets; scamper provides the ability to install a temporary firewall rule for some firewall types. Scamper runs on BSD, Linux, MacOS X, Solaris, and Windows systems; however, not all portions of the portability layer work on all systems.

Operation: scamper can run either as a one-shot measurement or as a daemon that is externally controlled. Scamper receives instructions in two ways. One, it accepts a list of IP addresses to probe, either in a file or on the command line itself, along with a measurement technique to use with each address specified on the command line. When all addresses have been probed, scamper has completed all required work and exits. The second way is to start scamper as a daemon and then connect to scamper's control socket and issue measurement instructions dynamically. This can be done either by specifying a measurement technique and a file with IP addresses to use, or by specifying individual measurement instructions interactively. The latter approach is powerful because it allows a macroscopic Internet measurement infrastructure to rapidly begin collecting data without implementing its own parallelised measurement tools and data collection system.

Event driven: there are no threads in scamper, so authors of new measurement techniques do not have to be concerned about reentrancy of their functions. To achieve parallelism, scamper uses non-blocking file descriptors in conjunction with the select system call. The main limitation with select is there is no guarantee that it will return exactly when the timeout expires. However, the finer resolution of the modern operating system scheduling clock has reduced the scale of this problem. Also, some techniques are self-clocking; for example, traceroute usually sends the next probe immediately after the last has been received, rather than waiting for a timeout.

Parallelism: scamper has two parameters that control its parallelism. The first is the minimum inter-packet transmit delay permitted, defined in packets per second (PPS). The default value of 20 means that scamper will send packets spaced 50ms apart. The second, a window parameter, defines the maximum number of active tasks permitted at any one time. The default value of zero means the window is unrestricted and the parallelism is defined solely by the PPS value. Scamper has been observed to probe at 1000 PPS on modest hardware, suggesting the event driven model scales well. Scamper aims to reach the desired rate by adding new tasks as required. As scamper can have multiple input sources supplying tasks concurrently, the user can specify the priority of each source. A priority value defines the ratio of new tasks it contributes overall in weighted round robin. If a source is not ready to supply a command and scamper has room in its probing budget, it obtains a new task from the next ready input source to maximise work done.

Stand-alone: as one design objective is to make scamper easy for volunteers to install and operate, scamper has no dependencies on external libraries or a configuration file, and has a small number of parameters configurable on the command line. These features make scamper easy to compile, install, and run. We considered using external libraries such as libpcap and libdnet, but not all operating systems contain a recent version of these libraries, and these libraries presently do not contain all required features.

4. MEASUREMENT TECHNIQUES

Traceroute: scamper began with a desire to conduct IPv6 traceroute in parallel to a large number of target addresses in support of CAIDA's macroscopic Internet topology discovery project [5]. The traceroute included in scamper is feature-rich: it supports IPv4 and IPv6; probe methods based on UDP, ICMP, and TCP, including Paris traceroute [12]; path MTU discovery (PMTUD) to infer the presence of tunnels [17]; a method to infer the hops in a path that do not send an ICMP packet too big message which is required for PMTUD to work [18]; and doubletree [19] to reduce redundant probing. It is optimised for macroscopic Internet topology discovery by halting if a sequence of unresponsive hops is encountered.

Ping: ping is useful to measure end-to-end delay and loss, search for responsive IP addresses, and classify the behaviour of hosts by examining how they respond to probes. In addition to the traditional ICMP echo method, scamper supports UDP, TCP, and TTL-limited probing, which can be used if directed ICMP echo probes do not obtain a response. Scamper includes the ability to spoof the source address of probes, as well as include IP options for record route and timestamps; these features are useful for implementing reverse traceroute [20].

MDA traceroute: scamper implements the multipath detection algorithm described in [21] to infer all interfaces visited between a source and destination in a per-flow load-balanced Internet path. It does this by deliberately varying the flow-identifier that a router may compute when load balancing. Probes with different flow-identifiers may take different paths and thus reveal different parts of the forward IP path. In addition to the ICMP and UDP methods originally implemented by Augustin *et. al* which vary the ICMP checksum and UDP destination port values, scamper implements a UDP method which varies the source port instead of the destination port so that the probes do not appear to be a port scan. This method also provides the ability to probe past a firewall that blocks UDP probes to ports above the usual range used by traceroute [22]. Scamper also implements TCP methods that vary the flow-id by changing the source or destination port, depending on the user's choice.

Alias resolution: scamper implements four techniques for inferring which IP addresses observed in the Internet topology are aliases. First, it implements Mercator probing [23] which infers aliases when a common source IP address is observed in ICMP port unreachable responses to probes sent to different destination IP addresses. Second, it implements Ally probing [24] which infers aliases by observing a sequence of increasing IP-ID values in response to probes interleaved to two different targets. Third, it implements RadarGun probing [25] where all candidates are tested simultaneously and aliases are then inferred by observing different IP addresses with the same IP-ID velocity. Finally, it implements a prefix scan method which infers a router's outgoing interface by finding an alias in the same subnet as the next interface in the forward path. As with the traceroute and ping implementations, scamper supports UDP, ICMP, and TCP probe methods. It also supports sending TTL-limited probes with a specific 5-tuple of values to solicit ICMP time exceeded messages from routers in a path; the 5-tuple can be obtained from the data recorded by scamper with traceroute data. This is useful for mapping router-level topologies when operators firewall probes directed at their routers, or do not announce prefixes used to number router interfaces.

Sting: scamper implements Savage's [15] TCP-based algorithm to infer one-way loss by making use of algorithms used by TCP receivers when they receive out-of-sequence packets. The technique is challenging to implement because a firewall rule must be inserted to prevent the host's operating system from interfering in the measurement by sending a TCP reset in response to a packet it does not expect. There is no standardised method across operating systems for an application to request particular packets be ignored. The original implementation of Sting no longer runs on a modern operating system due to the firewall interfaces changing substantially in the past ten years.

TBIT: scamper implements two of the techniques described in [16]: measurement of behaviour in response to an ICMP packet too big message, and measurement of behaviour in response to ECN negotiation and notification. As with Sting, a firewall rule is required with each measurement to prevent the host operating system from interfering. These techniques are well suited to being implemented in scamper because the probes are not time critical and fit with scamper's method of probing at a constant rate defined in PPS. However, sometimes bursts of packets are required, such as when ramping up a TCP connection through slow start so that the server response to a dropped packet can be measured. We are investigating methods to support parallelised measurement of this class of measurement.

5. EVALUATION OF SCALABILITY

This section demonstrates the performance of scamper measured by memory and CPU usage when run with different PPS rates on modest hardware. In particular, we explore the impact of increased parallelism on performance. We test performance using scamper's implementation of Paris traceroute with UDP probes [12]. Our experiment uses traceroute in the style of macroscopic Internet topology discovery [5]; traceroutes are launched to a randomly generated set of destination Internet addresses. This experiment tests multiple dimensions of scamper. Most traceroute measurements to randomly selected addresses are long-lived because most destinations are unresponsive to probes, requiring scamper to use memory to store information about tasks in progress. Even though most destinations are unresponsive, most probes are responded to by intermediate routers that send time exceeded messages. This requires scamper to efficiently use the CPU to receive and decode responses, determine the appropriate task that requires the response, store the response, and then take an appropriate action.

Our probing host is a Pentium 3 800Mhz with 128MB of RAM and a 100Mbps Ethernet interface, running FreeBSD 8.0. We compiled scamper with gcc's default settings on FreeBSD to build with compiler optimisations at level two. We instrumented scamper using the getrusage system call, recording the total amount of user and system CPU time consumed, as well as the maximum resident set size (RSS) which defines the maximum amount of RAM used by a process. We also recorded the percentage of CPU consumed by scamper at five second intervals because scamper is likely to be idle for a fixed amount of time towards the end of its workload while tasks timeout due to three consecutive unresponsive hops. We tested scamper at rates between 100 and 1000 PPS, in random order. For each probe rate, we used

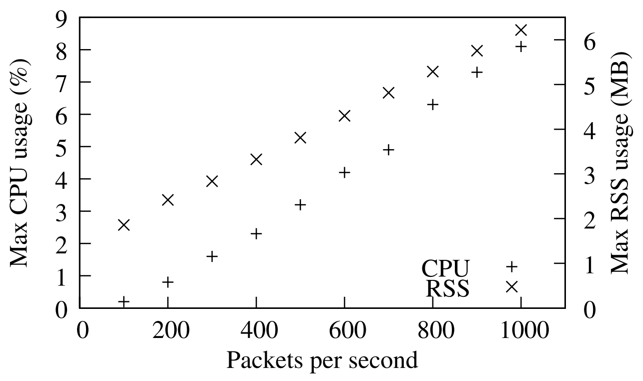

Figure 2: Maximum CPU and RSS usage observed for different packets per second rates. With traceroute, scamper's CPU and memory requirements grow linearly with probing speed, but neither requires significant computing resources.

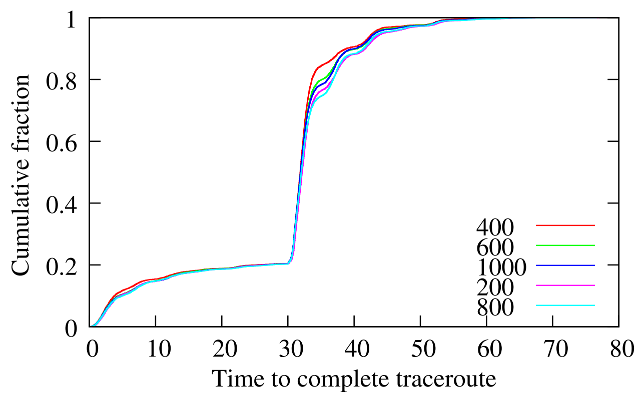

Figure 3: Time required to complete tasks at different PPS rates. The time elapsed for each traceroute is almost identical. The separation at 32 seconds is caused by some tasks waiting an additional timeout.

the same list of 10,000 randomly selected IP addresses from prefixes observed at Route Views [26]. We did not probe beyond 1000 PPS because of measurement etiquette considerations, as most probes in this experiment solicit a response from an intermediate router. The machine is otherwise idle.

CPU consumption: All of our tests consumed nearly the same total amount of CPU time regardless of PPS rate, between 25.3 seconds and 26.9 seconds. This can be explained by the use of efficient data structures that scale in $O(logn)$ and sharing sockets amongst tasks so that only a few sockets need to be monitored. The total CPU time consumed is greatest for the tests with the lowest PPS rates, because the scamper process accounts for more wall and therefore system CPU time. Figure 2 plots the maximum CPU usage observed for scamper when configured with different PPS rates. CPU requirements grow linearly with probing speed. This experiment is well within system capabilities of our Pentium 3 800Mhz, requiring 8.1% of the CPU at peak when probing at 1000pps.

Memory consumption: In order to meet a specified PPS rate, scamper adds new tasks as necessary, each requiring additional memory for maintaining state. Figure 2 also plots the maximum RSS reported by the operating system for each test. At 100pps, scamper requires 1.9MB at peak, and memory requirements grow linearly to 6.2MB at 1000pps. The memory and CPU requirements to run scamper are modest.

Effect on data collected: Overall, summary statistics for the data collected are almost identical for each PPS rate; 5.2% of destination were reachable, 10.0% received an ICMP destination unreachable code, 5.9% halted because of an apparent forwarding loop, and 78.9% halted after three consecutive unresponsive hops. In addition, the number of packets sent for all data sets is between 232k and 234k, and the number of IP-level links is between 29.2k and 29.3k. Figure 3 plots the time required to complete each traceroute for selected PPS values; the lines for other PPS rates fall between the PPS values that are plotted. For this workload, the distribution of time required to complete task is also almost identical for all PPS rates; however, 10% of tasks take up to five seconds more to complete in some experiments – the time that traceroute waits before timing out and sending another probe. There is no correlation between the probing rate and these variations; our experience with experiments at other times leads us to believe that the differences are due to network conditions at the time of this sample.

6. USING SCAMPER

As described in Section 3, scamper provides the ability to be run as a daemon and then controlled using a Unix domain socket or a TCP socket bound to the loop-back interface. As a simple example, consider a researcher who wishes to dynamically collect data about router-level connectivity available towards a set of destinations. To do so, the researcher requires the ability to collect the interface graph using MDA traceroute, determine the utility of probing each interface with TCP, ICMP, and UDP probes with ping to determine which probe types are able to solicit responses with incrementing IP-ID values, and then the ability to collapse the graph using an alias resolution technique such as RadarGun. The driver program the researcher would write to support this collection follows from the data requirements.

Briefly, the driver is responsible for remembering the list of targets to probe with MDA traceroute, the list of interfaces to probe with ping, and the utility of each ping method for soliciting an incrementing IP-ID so that repeated ping measurements are unnecessary. While there are targets to probe, the driver communicates to scamper the measurement tasks for it to traceroute and ping. As each measurement completes, scamper sends the results back to the driver across the control socket. The data arrives in a binary form, so to decode it the driver could use a socketpair, writing the binary data in one end and then reading the decoded measurements from the other using the provided API for doing so. At the same time, the driver can record the binary data to disk to archive the raw data for later use. When the list of targets to probe has become empty and all results are back from scamper, the final step is to provide scamper with a RadarGun probing specification containing the interfaces to probe paired with the appropriate probe method it should use, be it ICMP, UDP, or TCP. When the data collection is complete, the final step required is process the data collected to infer which interfaces are aliases, and then produce the router-level graph from it.

7. EXPERIENCES

Identifying IPv6 network problems in the dual-stack world: the first use of scamper was a 2004 study that compared the forward paths of IPv4 and IPv6 addresses believed to belong to the same host [17]. The goal was to use the data to find IPv6 paths that performed poorly compared to their IPv4 counterpart, and to provide operators with suggestions on where IPv6 routing could be improved. The work used scamper's one-shot measurement functionality; a list of IPv4 and IPv6 addresses were provided to scamper which then probed addresses in parallel until the list was completed. The main findings of this work were (1) only a small proportion of targets have a much larger delay with IPv6 than with IPv4, and (2) the impact of IPv6 tunnels, inferred by changes in Path-MTU, depends heavily on the upstream connectivity of the vantage point. With the IANA pool of remaining IPv4 addresses rapidly running out, it is important that the worst of the indirect IPv6 routes be identified and mechanisms created to make operators aware of them. Future work will look into automating data collection and procedures to communicate problems to operators.

CAIDA's macroscopic Internet topology project: scamper is used in CAIDA's macroscopic Internet topology discovery project to collect forward-IP path data on a continuous basis, beginning in September 2007. As of August 2010, there are 48 vantage points distributed across the globe, divided into three teams. Members of each team collectively probe a randomly generated address in all /24 prefixes routed on the Internet. Work is coordinated amongst vantage points using the Marinda tuple-space system [9]. An external measurement process monitors scamper's progress; when a unit of work is complete, a new set of random IP addresses is written to a file and passed to scamper using its control socket. Scamper is configured to probe at 100pps; a team of 12 vantage points requires approximately two and a half days to probe all routed /24 prefixes. The data collected is made available to researchers as the IPv4 Routed /24 Topology dataset [27]. There is substantial work to be done to translate the data into a router-level graph of the Internet using alias resolution, and the development of techniques to correctly filter AS-level links from it.

IPv6 AS-core poster: one of CAIDA's widely recognised visualisations is the IPv4 AS core poster showing the geographic connectivity and importance of ASes. In 2008, CAIDA had few vantage points with IPv6 connectivity with which to collect the raw IP topology data required, so volunteers were solicited on the NANOG mailing list. Each volunteer downloaded the scamper source code, compiled it, and then ran scamper using a supplied address list. Contributors from 53 different cities running varied operating systems supplied data, demonstrating scamper's portability and ease of volunteer use. The data was used to support a geographic comparison of the IPv4 and IPv6 AS-level graphs [28].

Traceroute probe method comparison: in 2008, we compared the utility of five different methods and found that ICMP-Paris traceroute reaches the most destinations and infers the most AS links when destinations are chosen randomly [29]. To collect the data required, we wrote a driver that connected to a running scamper process and issued a series of traceroute commands for each destination; the next method to use was chosen randomly by the driver, and it waited at least five seconds between traceroutes to any single destination. Using scamper's support for external drivers allowed us to focus on the data collection and analysis aspects of the work, rather than implementing our own complicated probing loops and algorithms.

Quantifying the pitfalls of traceroute: in 2009, researchers at Harbin Institute of Technology and UCLA examined the limitations of using traceroute data and the corresponding longest BGP prefix matches to infer AS connectivity [30]. To do so requires the ability to collect traceroutes from a vantage point where a BGP feed is also available. CAIDA's Ark topology project has three such vantage points; a fourth was created in UCLA by using scamper to collect traceroute data from their campus network where they were also able to procure a BGP feed.

8. RELATED WORK

Spring *et al.* built scriptroute with a goal of allowing implementations of measurement techniques to be portable and able to be used on a public general purpose measurement facility [6]. Both scriptroute and scamper include a portability layer so researchers can focus on the logic of their technique, and include implementations of common measurement techniques. Scriptroute provides researchers with a distributed set of machines from which to run experiments; with scamper, we focused on building a parallelised packet-prober that can be used easily in both measurement infrastructure and stand-alone contexts.

fping [31], hping [32], and nmap [33] have portable and parallelised implementations of ping, traceroute, and network security tests respectively, allowing networks to be rapidly tested. Scamper is built with the Internet researcher in mind; it provides an extensible file-format that records the detail of a measurement to provide a researcher with sound data, and provides flexibility in the control of the measurement process making it suited for use in Internet measurement infrastructure.

9. CONCLUSION

Internet researchers are faced with many challenges when building and designing experiments; overcoming technical limitations of operating systems, recording results and data in a way that allows sound analysis to take place, parallelising their implementation so it can scale to meet the size of the Internet, and finding enough vantage points to be confident their results are representative of the Internet. Scamper provides a flexible and reusable packet-probing architecture that allows researchers to focus on scientific experiments rather than building accurate and scalable instrumentation. Scamper's architecture has been shown to be useful to a community-oriented network measurement infrastructure, as it is currently used in CAIDA's Archipelago system [9]. The source code to scamper is freely available at `http://www.wand.net.nz/scamper/`.

Acknowledgements

We thank WIDE and CAIDA for funding and supporting the initial development of scamper. This work is currently supported by New Zealand Foundation for Research Science and Technology (FRST) contract UOWX0705.

10. REFERENCES

[1] David Mills. Internet delay experiments. RFC 889, December 1983.

[2] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An architecture for large-scale Internet measurement. *IEEE Communications Magazine*, 36(8):48–54, 1998.

[3] Sunil Kalidindi and Matthew J. Zekauskas. Surveyor: An infrastructure for Internet performance measurements. In *INET'99*, San Jose, CA, June 1999.

[4] Tony McGregor and Hans-Werner Braun. Balancing cost and utility in active monitoring: The AMP example. In *INET 2000*, Yokohama, Japan, July 2000.

[5] Bradley Huffaker, Daniel Plummer, David Moore, and k claffy. Topology discovery by active probing. In *SAINT 2002*, pages 90–96, Nara City, Japan, January 2002.

[6] Neil Spring, David Wetherall, and Tom Anderson. Scriptroute: A public Internet measurement facility. In *USITS '03*, pages 225–238, Seattle, WA, March 2003.

[7] Yavul Shavitt and Eran Shir. DIMES: let the Internet measure itself. *Computer Communication Review*, 35(5):71–74, 2005.

[8] Harsha Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: An information plane for distributed services. In *OSDI '06*, pages 367–380, Seattle, WA, November 2006.

[9] Young Hyun. Archipelago measurement infrastructure. `http://www.caida.org/projects/ark/`.

[10] kc claffy, Mark Crovella, Timur Friedman, Colleen Shannon, and Neil Spring. Community-oriented network measurement infrastructure (CONMI) workshop report. *ACM/SIGCOMM Computer Communication Review*, 36(2):41–48, April 2006.

[11] Van Jacobson. traceroute. `ftp://ftp.ee.lbl.gov/traceroute.tar.gz`.

[12] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. Avoiding traceroute anomalies with Paris traceroute. In *IMC '06*, pages 153–158, Rio de Janeiro, Brazil, October 2006.

[13] Mark Allman and Vern Paxson. A reactive measurement framework. In *PAM 2008*, pages 92–101, Cleveland, OH, April 2008.

[14] Young Hyun. rb-wartslib: ruby warts library. `http://rb-wartslib.rubyforge.org/`.

[15] Stefan Savage. Sting: a TCP-based network measurement tool. In *USITS '99*, pages 71–79, Boulder, CO, October 1999.

[16] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the Internet. *Computer Communication Review*, 35(2):37–52, April 2005.

[17] Kenjiro Cho, Matthew Luckie, and Bradley Huffaker. Identifying IPv6 network problems in the dual-stack world. In *ACM SIGCOMM workshop on Network Troubleshooting*, pages 283–288, Portland, OR, August 2004.

[18] Matthew Luckie, Kenjiro Cho, and Bill Owens. Inferring and debugging path MTU discovery failures. In *IMC '05*, pages 193–198, Berkeley, CA, October 2005.

[19] Benoit Donnet, Timur Friedman, and Mark Crovella. Improved algorithms for network topology discovery. In *PAM 2005*, pages 149–162, Boston, MA, March 2005.

[20] Ethan Katz-Bassett, Harsha V. Madhyastha, Vijay Kumar Adhikari, Colin Scott, Justine Sherry, Peter van Wesep, Thomas Anderson, and Arvind Krishnamurthy. Reverse traceroute. In *NSDI '10*, pages 219–234, San Jose, CA, April 2010.

[21] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring load-balanced paths in the Internet. In *IMC '07*, pages 149–160, San Diego, CA, October 2007.

[22] R-fx Networks. Advanced policy firewall (APF). `http://www.r-fx.ca/downloads/apf-0.9.6-3.tar.gz`.

[23] Ramesh Govindan and Hongsuda Tangmunarunkit. Heuristics for Internet map discovery. In *INFOCOM*, pages 1371–1380, Tel-Aviv, Israel, March 2000.

[24] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *SIGCOMM '02*, pages 133–145, Pittsburgh, PA, August 2002.

[25] Adam Bender, Rob Sherwood, and Neil Spring. Fixing Ally's growing pains with velocity modeling. In *IMC '08*, pages 337–342, Vouliagmeni, Greece, October 2008.

[26] University of Oregon. Route Views. `http://www.routeviews.org/`.

[27] Young Hyun, Bradley Huffaker, Dan Andersen, Emile Aben, Matthew Luckie, and kc claffy. The CAIDA IPv4 routed /24 topology dataset. `http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml`.

[28] CAIDA. Visualizing IPv6 AS-level internet topology 2008. `http://www.caida.org/research/topology/as_core_network/ipv6.xml`.

[29] Matthew Luckie, Young Hyun, and Brad Huffaker. Traceroute probe method and forward IP path inference. In *IMC '08*, pages 311–323, Vouliagmeni, Greece, October 2008.

[30] Yu Zhang, Ricardo Oliveira, Hongli Zhang, and Lixia Zhang. Quantifying the pitfalls of traceroute in AS connectivity inference. In *PAM '10*, Zurich, Switzerland, April 2010.

[31] fping - a program to ping hosts in parallel. `http://fping.sourceforge.net/`.

[32] hping - active network security tool. `http://www.hping.org/`.

[33] Gordon Lyon. nmap. `http://nmap.org/`.