A Service-Oriented Measurement Infrastructure for Cloud Computing Environments

Gregg Hamilton, Dimitrios Pezaros
School of Computing Science
University of Glasgow
Glasgow, UK
g.hamilton.3@research.gla.ac.uk, dimitrios.pezaros@glasgow.ac.uk

***Abstract*—With an ever-increasing number of services being moved to cloud computing platforms, it is essential for data centre operators to quickly identify any problems in their network in order to ensure optimal performance and provide guidance for when actions such as migration of computing instances are required. This paper introduces a distributed measurement infrastructure using low-cost in-line link measurements to reveal the true traffic-perceived service quality in short timescales. In-line measurement is highly relevant for data centres as it allows an accurate measure of important network services without the problem of injecting artificial traffic into a network with high contention for resources. The infrastructure presented allows real-time monitoring and visualisation of the state of the network, which can be used to inform virtual machine migration/instantiation events. We show that the in-line measurements used by the infrastructure have negligible impact on network traffic and the hosts conducting the measurements. The low overhead of in-line measurements potentially enables the infrastructure to function as an always-on measurement and management platform.**

***Keywords*-Measurement, network monitoring, network performance, distributed systems, cloud computing**

I. Introduction

Cloud computing has become prominent in recent years as a way for allowing end users to offload large processing tasks to data centres. With the volatile workloads such on-demand systems are required to handle network congestion can easily occur on links connecting hosts, requiring action to maintain the performance of particular services communicating on those links.

Anecdotal evidence suggests that regional cloud data centre operators typically upgrade their infrastructure only when they reach 80% utilisation. This is often because cloud operators cannot afford ISP-like over-provisioning of resources and require a better return on investment. If the network could be efficiently monitored problems could be quickly detected, enabling remedial action, such as migration of instances, and longer timescales before infrastructure upgrades are required.

This paper introduces a low-cost, real-time measurement infrastructure using in-line measurements to address the issue of informative always-on real-time measurements of actual network traffic at short timescales.

II. Background

A. Cloud Computing

Cloud computing services, such as Amazon's EC2, typically give users access to a computing *instance*, or a collection of instances, on which to perform computations and charge for the time that the instance is being used. Users can often scale the required number of instances up or down dynamically to either increase performance or reduce the overall running time of a distributed set of tasks.

Within cloud computing infrastructures, instance placement and migration must be taken into consideration. Placement is the act of deploying a virtual machine (VM) instance and migration is the act of moving an instance from one deployment location to another, while in a running state. New instances must be carefully placed such that they can achieve expected performance levels without the activity of existing instances degrading that performance. Similarly, if the performance of an instance is becoming severely degraded it is desirable to migrate that instance elsewhere. The measurement infrastructure presented in this paper can aid in the placement and migration process by allowing network operators to make informed decisions based upon the real-time network performance.

B. Network Measurement

In an environment such as a data centre there could be tens of thousands of interconnected servers, each hosting several VMs. Each of the VMs may be competing with others for already over-subscribed network resources. If this is the case, it is sometimes desirable to migrate VM instances to another server where there will be less contention. To handle the measurement of such large-scale distributed environments, suitable infrastructures must be put in place that would take into account system and network resource contention.

Numerous types of network measurement techniques exist such as passive [1][2], active [3][4] and hybrid [5] measurements. However, active measurements are only as good as the type of traffic that they mimic, while passive measurements can be costly to process and are often only of use for analysis of network problems after they have occurred. In a cloud environment, many network services may need real-time measurement results. The in-line measurements used in this paper address this with the ability to measure any type of traffic, allowing a diverse picture of actual network traffic performance.

III. An Always-On Network Measurement Infrastructure

The main goal of this paper is to introduce a distributed network measurement infrastructure, suitable for always-on low-cost measurements which reveal the real performance of traffic in short timescales. There exist previous attempts at monitoring large-scale networks, such as ISP networks [1][3]. Cloud data centres, often composed of tens of thousands of servers, pose similar problems.

Work on cloud environments has looked at monitoring individual computing instances or clusters of instances in cloud data centres [6], but not the links connecting instances. In a data centre the servers may be operating upon large distributed tasks, e.g., *MapReduce* jobs, where network performance is a serious consideration. To address this, we implemented a distributed measurement infrastructure capable of using in-line measurements [7].

In-line network measurements are low-overhead measurements that are piggybacked within traffic already destined for the network. This is achieved with a source module that attaches measurement data to packets and a destination module that retrieves measurement data from packets. For high-performance operation, the modules operate within the Linux kernel, necessitating their implementation in C. The in-line measurement modules are discussed fully in Section IV-A.

Our infrastructure is intended to make deployment and operation of the infrastructure itself relatively painless. The system is focused around a network operations centre (NOC) which is used to coordinate measurements and aggregate and visualise measurement data.

A. Implementation and Deployment

The Java programming language provides remote method invocation (RMI) for method calls between hosts across a network, and the Java Native Interface (JNI) for interaction with native C code to allow communication with the Linux kernel modules, making Java a suitable choice for this infrastructure.

In-line measurements operate in kernel space, requiring modifications to the Linux kernel. Data centre operators can avoid patching individual servers by deploying pre-configured custom operating system images to collections of machines, along with the associated Java components.

B. Interface

The interface presented to a user is graphical and consists of several distinct aspects, as shown in Fig 1. The main window of the interface presents the user with a module configuration panel, a network topology panel, and a real-time measurement chart panel. The primary function of the interface is to easily visualise the state of the network between the measurement hosts. As measurement traces are transmitted back to the NOC, charts are displayed for one-way delay or loss between two hosts, enabling an at-a-glance notification of the current state of the network. The real-time visualisation of network traffic performance can easily allow for identification of a problem when it occurs, allowing an operator to make an informed decision to manually migrate a set of instances, or avoid placing any new instances in a particular area within the data centre that may be experiencing network troubles. As the number of links between end-hosts monitored by the infrastructure increases, intermediate measurement controllers and aggregators could be introduced, which would allow for offloading of high quantities of network traffic and data processing from the NOC itself.

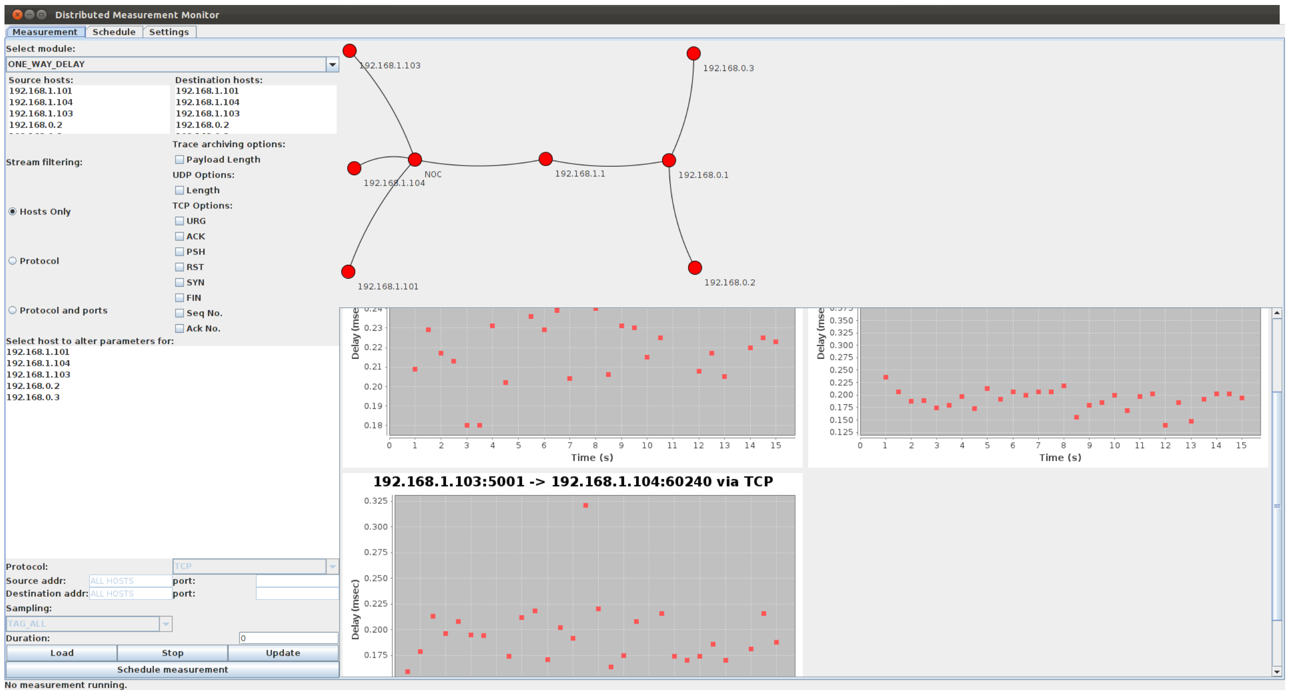
The interface also attempts to provide a topology visualisation, which can be of great value in a complex network setup such as that of a data centre. To achieve the topology visualisation, the NOC coordinates traceroute measurements that are remotely initiated between end-hosts and the resulting outputs are used to construct a topology graph at the NOC. However, if traceroute is unable to identify all hosts during an execution, the unknown hops are omitted from the topology visualisation.

C. Configuration

The measurement infrastructure primarily allows the ability to conduct real-time network measurements and view instant results of those measurements. Java RMI enables a NOC to insert a module on a remote host to begin network measurement or remove a module to terminate a measurement session. Measurement sessions can also be scheduled for future execution among a set of hosts, which can be exploited for monitoring the network at various points throughout the day for operators to get regular snapshots of the network performance. The setting-up of a time for a future measurement session, and the schedule of planned sessions, are shown in Fig 2 and Fig 3, respectively.

Sampling schemes can be configured for individual hosts, allowing monitoring to be adapted for a subset of traffic of interest and limiting the frequency with which measurements are piggybacked.

To allow for analysis of a measurement session, measurement traces are transmitted back to the NOC. The traces stored and transmitted are small and configurable, often consisting of only host and port information along with the appropriate measurement information, such as delay values. When traces are transmitted back to the NOC they are reconstructed back into flows at user-defined levels of traffic granularity, e.g., at host level only or at host-and-port level.


Select module: ONE_WAY_DELAY

Source hosts:
192.168.1.101
192.168.1.104
192.168.1.103
192.168.0.2

Destination hosts:
192.168.1.101
192.168.1.104
192.168.1.103
192.168.0.2

Stream filtering: [✓ Hosts Only] Protocol Protocol and ports

Trace archiving options: Payload Length
UDP Options: Length
TCP Options: URG ACK PSH RST SYN FIN Seq No. Ack No.

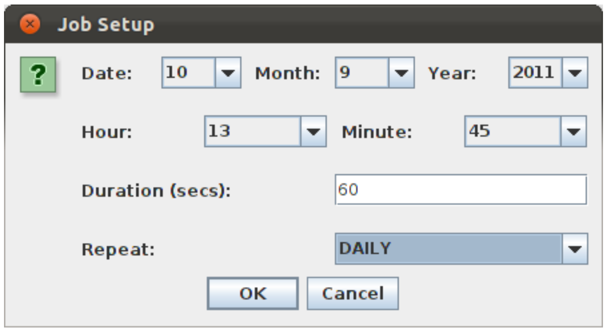
Select host to alter parameters for:
192.168.1.101
192.168.1.104
192.168.1.103
192.168.0.2
192.168.0.3

Protocol: TCP
Source addr: ALL HOSTS port:
Destination addr: ALL HOSTS port:
Sampling: TAG_ALL
Duration: 0

192.168.1.103:5001 -> 192.168.1.104:60240 via TCP

No measurement running.

Figure 1. The distributed measurement interface

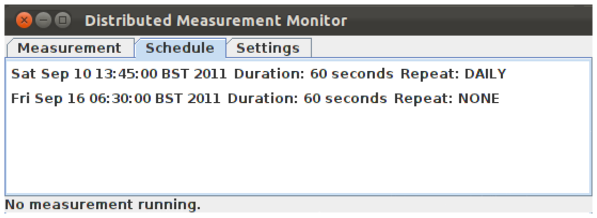

Date: 10 Month: 9 Year: 2011

Hour: 13 Minute: 45

Duration (secs): 60

Repeat: DAILY

Figure 2. Configuration of a future measurement session


Sat Sep 10 13:45:00 BST 2011 Duration: 60 seconds Repeat: DAILY
Fri Sep 16 06:30:00 BST 2011 Duration: 60 seconds Repeat: NONE

No measurement running.

Figure 3. Future scheduled measurements

This is primarily useful for per-flow visualisation in the analysis of traffic performance.

D. *Integration with Measurement Modules*

The in-line measurement modules operate within the Linux kernel. The benefit of having the modules operate

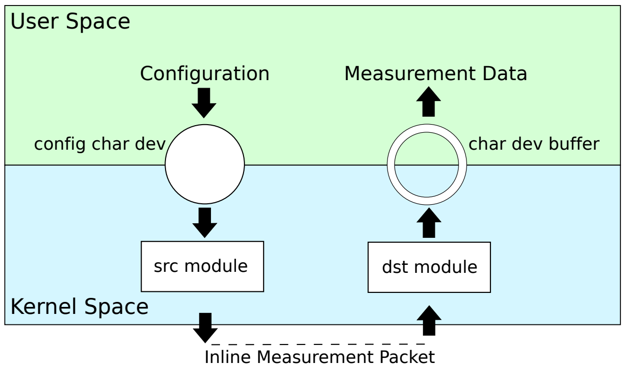


Figure 4. Operation of measurement modules

from within the kernel is more efficient piggybacking of measurement data by modifying packets directly within the kernel. Communication with user space applications is via Linux character devices, allowing the measurement infrastructure to configure measurement modules and retrieve measurement data.

In-line measurements are carried as a pseudo-layer between the IP and transport layers of a packet. When a packet containing measurement data is received at a host, the data is stripped from the packet and passed to a character device containing a circular buffer. When data is read from the character device, it is copied from kernel space into user space memory. The communication processes of the modules are illustrated in Fig 4.

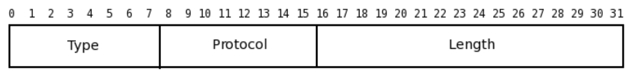


Figure 5. Measurement data header

JNI code reads raw measurement data from native character devices into a Java-based parser that reconstructs the measurement results, and associated identifying headers from the IP and transport layers, into Java objects that can be used within the infrastructure, e.g., for measurement visualisation purposes.

IV. In-line Measurements

In-line measurement techniques are currently implemented natively in IPv6 using extension headers. However, IPv6 has not yet come to prominence, leaving IPv4 as the dominant Internet protocol. To achieve in-line measurements in IPv4, a new layer has to be introduced into the IPv4 stack. The implementation of in-line measurements for IPv4 traffic is discussed in this section.

A. Measurement Injection

In order to piggyback measurement data we inject it as a pseudo-layer between the IP layer and the transport layer, as this makes the injection mechanism suitable for all Internet traffic. Like other TCP/IP layers, it requires a header to identify its presence and properties. The header used to identify in-lined data is shown in Fig 5.

As the in-lining mechanism is intended to allow future expansion the header incorporates a measurement type field. The header also contains a next protocol field to identify the transport layer protocol. The next protocol field in the IP header is set to 253, a value reserved for experiments as outlined in *RFC 3692*, and the original next protocol value is copied into the in-line header. Finally, the header also has a length field so that measurement data may be successfully removed.

While insertion of measurement data between the IP and transport layers should allow for routing to operate as normal, strictly configured routers could drop the packet due to an unrecognised *next header* value in the IP header. To overcome this, we perform IP-in-IP encapsulation on the packet, as defined in *RFC 2003*, to tunnel measurement packets through a network.

The injection mechanism discussed here requires changes only to the TCP/IP stacks of end-hosts conducting measurements and no modification of any systems performing intermediate routing.

B. Measurement Modules

To illustrate the use of the in-lining mechanism, two types of network measurement were implemented: one-way delay and one-way loss. The rationale and benefit of implementing such measurements in an in-lined nature is discussed in this section.

One-way delay: Measuring one-way delay across network links between hosts requires high resolution timestamps for accurate measurements. Our measurement mechanism allows for an in-line one-way delay measurement module that injects source and destination timestamps as close to the link layer as possible by implementation within the Linux kernel.

One-way loss: Packet loss typically goes hand-in-hand with link delay measurements. If a link has a high delay value it will often indicate congestion. Such congestion often leads to an increased loss of packets which can lead to further congestion as packets are re-transmitted.

C. Configurable Measurement Control

Although in-line measurements give significantly less overhead than techniques that inject extra traffic into the network, it may not be of interest to measure and analyse all the traffic in a network. In order to facilitate a variety of targeted measurement capabilities, we have implemented a configurable packet sampling system. This comes with the ability to rate-limit the measurement data that is sent in-line with packets. A *1-in-N packets* and *1-per-N milliseconds* sampling scheme are offered.

D. Implementation

The measurement mechanism was implemented by adding hooks to version 2.6.30 of the Linux kernel and implementing the one-way delay and one-way loss measurements as loadable kernel modules (LKMs). The hooks were arrays of pointers to functions, allowing the measurement modules to plug processing functions into them, so that they may insert or retrieve measurement data.

In order for the insertion or retrieval of measurement data to take place as close to the link layer as possible, handling code is placed in each of the kernel `ip_finish_output2()` and `ip_rcv()` functions, which loops over the hooks described above. These are the last function to handle a packet before it is dispatched to the network hardware and the first to handle a packet after it is received by the network hardware, respectively.

We reserve space for measurement data and the encapsulation header by clamping the MSS in TCP packets. Once a packet with enough spare space has been identified the kernel socket buffer is modified accordingly to insert the measurement data and its header.

Before the outgoing packet is handed back to the network, it is wrapped in an outer IP header to ensure safe passage through intermediate routers, as described above, which is then removed when the packet reaches its destination.

As the mechanism operates within routines in kernel space, communication must take place between kernel space and user space, facilitated by character devices for measurement configuration and retrieval of measurement data.

A second implementation of the measurement mechanism was also developed using the Click Modular Router to compare the performance of an extensible router framework to native kernel code [8]. Click is a router framework which consists of a collection of self-contained modules that each perform a single task upon a packet, such as applying an IP header. Multiple Click modules can be combined in a *configuration* and run as a software router.

V. Evaluation

In order to evaluate the impact of the infrastructure upon a network, a set of network throughput experiments was conducted on the underlying in-line measurements used within it. These experiments were conducted on a 2.4GHz Pentium 4 with a 100 Mbit/s NIC and an AMD Athlon 64 3500+ with a 100 Mbit/s NIC. The two systems were connected via a 100 Mbit/s switch.

A. Throughput

The throughput tests were first run to gain an understanding of the effect the measurement mechanism has on network performance. In the tests, for comparison, the measurement mechanism was run against the native Linux kernel and the Click prototype. The experiment was run on four setups in total:

- A vanilla Linux kernel
- A basic Click IP router setup
- A Click IP router with in-line measurement modules
- A modified Linux kernel using in-line measurements

Linux kernel 2.6.24.7 was used on both systems, due to the latest version of the Click kernel patch available. One-way delay was the measurement module used. The *Iperf* network testing tool was used to conduct the throughput experiments on packet sizes of 150, 250, 500, 1000 and 1500 bytes.

Each test was run for 10 seconds and five runs were conducted for each packet size, with the result being the average of the five runs. The results of the throughput tests are shown in Mbit/s in Fig 6.

As the figures show, the in-line measurement mechanisms have little effect on the throughput when large packet sizes of 1000 or 1500 bytes are transmitted. Packets of this size, which are close to the Ethernet MTU, are the most prevalent in transfers in the Internet, so in real world usage adding in-line measurements to packets around this size should have a negligible effect on network throughput.

However, as packet size decreases, the performance gap widens. As more packets can be sent on a link per time unit when using smaller packets, the lower throughput is likely caused by the per-packet processing time. It is worth noting that although sending 150 byte packets with in-line measurements only achieves a throughput of around

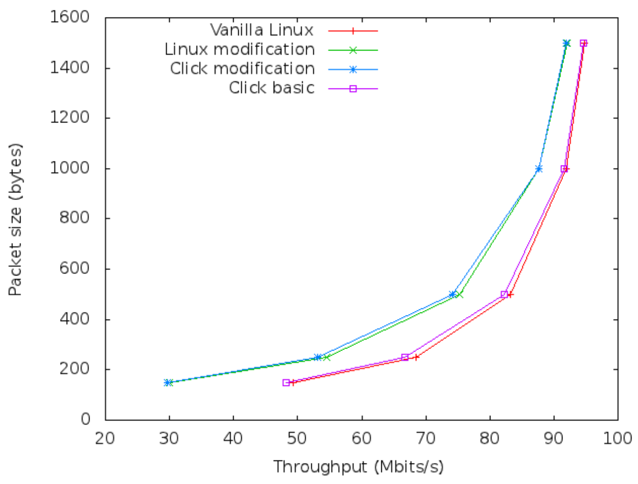


Figure 6. Throughput in Mbit/s

Table I
PROCESSOR UTILISATION BY LINUX AND CLICK IMPLEMENTATIONS

	Processor utilisation			
	Linux		Click	
Source module	one_way_delay_src	2.0%	*Total:*	*19.2%*
	ipv4_encapsulation	4.8%		
	Total:	*6.8%*		
Destination module	one_way_delay_dst	0.65%	*Total:*	*4.5%*
	ipv4_decapsulation	0.43%		
	Total:	*1.8%*		

30 Mbit/s, a vanilla Linux kernel could still only achieve a throughput of 49 Mbit/s. However, we do not advise instrumenting minimum-sized packets.

B. Profiling

To evaluate the performance impact on individual hosts, another 10 second throughput experiment using 150-byte MTU packets was run, this time using the *OProfile* [9] system profiler to measure processor utilisation on the hosts. A 150-byte MTU was chosen as the application-level minimum allowed by the experimental setup, but is close to the theoretical minimum of 64 bytes. OProfile instrumented the respective unhalted state counter on each processor, with a counter limit of 10^5. The results are presented in Table I.

While Click may achieve a comparable throughput to Linux, the Click architecture has a much higher processing overhead, requiring roughly two and a half times the processing power. An observation of interest is that the destination module has a significantly lower overhead in both implementations. This is down to the potential allocation of extra packet headroom and the resulting copying of socket buffers that may occur in the source module.

These evaluations have shown that, in the common case of near MTU-sized packets, the measurement mechanism operates with a very low overhead on both network performance and processor utilisation. The comparison of the

Click prototype and Linux implementation has shown that while they may achieve comparable throughput, Click does so at a substantial cost in processor usage.

VI. Related Work

The work of most relevance to this paper are in-line measurements for IPv6 [7]. There have also been studies into providing frameworks for programmable network measurements [10][11].

Network measurement infrastructures for large-scale networks in general have been widely studied in the literature. However, they are often based upon costly active measurements [4], or are based on passive measurement techniques [1]. There have been few attempts focused on measurement in cloud data centres [6]. However, such work is limited to exploring the monitoring of the performance of individual instances and clusters of instances, rather than the critical networks required for inter-instance communication in cloud data centres.

In terms of adding extra functionality to the networking stack, as the measurements in this paper introduce, there is the *Click Modular Router*, which is a module-based routing framework that allows the quick construction of custom routers [8]. There are also *SILO*, which attempts to apply cross-layering to the networking stack [12] and *NetServ*, which looks at adding services at the core of the network [13].

VII. Conclusion

This paper has presented a low-cost in-line measurement infrastructure for data centres. The infrastructure has been designed to allow the measurement of data centre networks in short timescales, revealing the actual performance of any traffic type supported by the data centre in real-time. Measurement sessions can be reconfigured in real-time and future sessions can be scheduled, while charts of the performance of network traffic can be displayed. The information presented by the measurement infrastructure can be leveraged to make informed decisions on when to take action to maintain network performance, such as migrating or instantiating new VM instances.

An in-line measurement implementation for IPv4 has been provided for native Linux as well as for the Click Modular Router framework, and throughput experiments have demonstrated that with the typical near MTU-sized packets in the Internet there is little instrumentation overhead, even at full line speeds, while moderate sampling would further reduce any overhead to virtually zero.

The mechanism introduces extensibility to the TCP/IP stack, with the potential for further control and management applications, e.g., performance-based routing updates for link-state protocols like OSPF. New in-line data injection applications can be realised merely by implementing a new control data structure in a kernel module and loading it onto a running kernel.

References

[1] D. Antoniades, P. Trimintzios, M. Polychronakis, S. Ubik, A. Papadogiannakis, and V. Smotlacha, "Lobster: a European platform for passive network traffic monitoring," in *Proc. TridentCom '08*, 2008, pp. 8:1–8:10.

[2] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, "CSAMP: a system for network-wide flow monitoring," in *Proc. 5th USENIX Symp. on Networked Systems Design and Implementation (NSDI '08)*, 2008, pp. 233–246.

[3] M. Luckie, "Scamper: a scalable and extensible packet prober for active measurement of the internet," in *Proc. 10th Annual Conference on Internet Measurement (IMC '10)*, 2010, pp. 239–245.

[4] H. H. Song, L. Qiu, and Y. Zhang, "NetQuest: a flexible framework for large-scale network measurement," in *Proc. Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '06)*, 2006, pp. 121–132.

[5] P. Papageorge, J. McCann, and M. Hicks, "Passive aggressive measurement with MGRP," in *Proc. ACM SIGCOMM '09*, 2009, pp. 279–290.

[6] S. De Chaves, R. Uriarte, and C. Westphall, "Toward an architecture for monitoring private clouds," *Communications Magazine, IEEE*, vol. 49, no. 12, pp. 130–137, December 2011.

[7] D. Pezaros, M. Hoerdt, and D. Hutchison, "Low-overhead end-to-end performance measurement for next generation networks," *IEEE Transactions on Network and Service Management*, vol. 8, no. 1, pp. 1–14, March 2011.

[8] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, pp. 263–297, August 2000.

[9] "Oprofile system profiler." [Online]. Available: http://oprofile.sourceforge.net/

[10] J. Sommers, P. Barford, and M. Crovella, "Router primitives for programmable active measurement," in *Proc. 2nd ACM SIGCOMM Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO '09)*, 2009, pp. 13–18.

[11] L. Yuan, C.-N. Chuah, and P. Mohapatra, "ProgME: towards programmable network measurement," *IEEE/ACM Transactions on Networking*, vol. 19, pp. 115–128, February 2011.

[12] R. Dutta, G. Rouskas, I. Baldine, A. Bragg, and D. Stevenson, "The silo architecture for services integration, control, and optimization for the future internet," in *Proc. IEEE International Conference on Communications (ICC '07)*, June 2007, pp. 1899–1904.

[13] S. Srinivasan, J. Lee, E. Liu, M. Kester, H. Schulzrinne, V. Hilt, S. Seetharaman, and A. Khan, "NetServ: dynamically deploying in-network services," in *Proc. Workshop on Re-architecting the Internet (ReArch '09)*, 2009, pp. 37–42.