Automating Network Monitoring on Experimental Testbeds

Michael Golightly
Princeton University
mgolight@gmail.com

Jack Brassil
HP Laboratories
jack.brassil@hp.com

Abstract

Despite experimental testbeds' rapid growth and continued strong demand by researchers, the power of testbeds can be further increased by providing additional tools to help experimenters instrument their experiments. Experimenters with improved instrumentation support can deepen their understanding of experiment operation, and have an easier task of generating high quality datasets to share with the community.

We introduce a prototype tool that automatically deploys an instrumentation overlay on an existing testbed experiment. *Netflowize* modifies instantiated experiments to collect experiment-wide flow statistics. The resources consumed by the flow collection process are specified by the experimenter. NetFlow records are widely used by the networking and security research communities for tasks ranging from traffic engineering to detecting anomalous behaviors associated with zero-day attacks. We discuss tool design and implementation, present usage examples, and highlight the many challenges of auto-deploying an experiment-wide monitoring infrastructure.

1 Introduction

Cyber security research continues to be hampered by the lack of quality data sets available to the research community. One factor contributing to this scarcity is the difficulty experimenters face in collecting and preserving useful data. Yet experimental testbeds offer relatively few services specifically targeted at instrumenting experiments.

In this paper we outline the design, functionality, and performance characteristics of Netflowize, an automated deployment tool for experimental network monitoring. The tool was developed to automatically collect *netflow* [1] information for any Emulab [2] experiment's network topology. NetFlow is a powerful flow monitoring tool heavily used by large-scale network administrators, but has been relatively lightly used by *experimenters* in testbed settings. The ability to export NetFlow data is a native capability on all but the least expensive IP routers and L2/3 switches, and the tools available to process collected data are numerous.

Though conceived as a general purpose network measurement platform with applications to traffic engineering, accounting and billing, NetFlow has been embraced by the network security research community. Some recent applications of NetFlow include studies in network forensics [3], botnet discovery [4], and incident detection; many other security research applications are considered at conferences such as CERT's *FloCon* [5]. Though NetFlow itself is of immediate interest to many experimenters, our extensible deployment framework was designed to generalize beyond NetFlow to support other monitoring tools. Hence what we learned about automated deployment is of general interest to readers considering automated monitoring of experiments using various other tools.

But deploying such network monitoring tools effectively – consuming minimal experiment resources and experiment behavior – takes fairly sophisticated users deeply familiar with each individual tool, and often experience with implications of instrumentation design choices. While many experimenters could benefit from enhancing instrumentation of their experiment, only some choose to. Yet we envision future testbeds with thousands of physical servers each with tens or hundreds of virtual machines, and in this setting individual experiment size will likely grow enormously. Automated deployment promises to be a tractable approach to monitor

such large-scale experiments.

In this paper we highlight the challenges faced in developing automated testbed instrumentation though the lens of netflowize development. In Sections 2 & 3 we introduce our motivation, and briefly review background on NetFlow. Section 4 describes our design goals, and details our implementation, including our approaches to the tradeoff between system resources and NetFlow probe and collector placement. The next section discusses some simple application examples. The final section presents our conclusions, and argues that automated deployment tools will be valuable in diverse settings beyond experimental testbeds, including commercial public clouds.

2 Methodology

Despite the wide acceptance and success of experimental testbeds, services for instrumenting experiments have developed slowly. We believe that lack of instrumentation support is a key contributing factor to the rarity of useful data sets collected by – and available to – the security and networking research communities. Yet instrumenting individual experiments of unknown type and purpose seems possibly beyond what a testbed can hope to provide. In the face of the additional complexity and cost, experimental testbed operators have provided some limited hardware and software tools (e.g., Endace DAG capture hardware [6]), but have largely left the experimenter to use standard platforms and create measurement systems for themselves.

In this paper we challenge testbed operators to sharpen their focus on offering services that enable the experimenter to deploy an experiment-specific instrumentation overlay. To investigate the feasibility and challenges of automated instrumentation deployment, we set out to build a prototype based on *netflow*. Creating a NetFlow overlay manually is not out-of-reach for many sophisticated users, but is less likely to be considered by certain testbed users such as students. The tool, which we believe will provide insights to other tool developers, has been made publicly available.

Though we have chosen to deploy NetFlow because of its immediate relevance to experimenters in both the network systems and network security research communities, we have striven to create an extensible framework which is largely independent of the underlying measurement tool deployed. In particular, we have made no changes to the open source NetFlow probes and collectors we deploy automatically. Hence, and tool that can operate in a client-server (or sensor-collector) mode can in principal be deployed with our framework. In particular, we imagine it to be straightforward to use other software-based measurement systems such as *sflow* and packet sniffers such as *tcpdump*.

3 Background

The NetFlow protocol was originally developed by Cisco Systems to collect flow information for various accounting purposes. Versions 5 and 9 are the most widely used formats today. V9 is template-based, allowing it to be more extensible and easily adaptable to recording additional flow information. The netflowize tool currently supports only the v5 protocol, though extending the framework to v9 is straightforward. A NetFlow v5 packet payload consists of a 24 byte header followed by at least one 48 byte flow record.

There are two main components to a NetFlow deployment, the probe and collector. A probe monitors traffic on a network device and generates NetFlow records that are sent to a collector. NetFlow probes are natively supported on most routers and many enterprise class L2-3 ethernet switches, and are usually configured through a CLI that allows specific interfaces to be monitored and filters to be constructed to monitor specific flows. Software-based NetFlow probes, collectors, visualizers, etc, are available with packages such as *flow-tools* [21] and *SiLK* [22] that can be run on most Unix-based systems.

A network flow is informally defined to be a unidirectional sequence of packets with some logical association (e.g., those packets belonging to a TCP connection). More precisely, a widely accepted definition is a sequence of packets whose headers match a specified n-tuple observed during an interval of time at a single point (e.g., router egress link). An example of such a tuple might be

```
< src IP, dest IP, src port, dest port, IP
protocol, IP Type of Service>
```

Each newly observed flow triggers creation of a 48-byte NetFlow record; NetFlow v5 records contain the following fields:
By analyzing flow data at multiple vantage points, an experiment-wide picture of traffic flow and traffic volume can be built.

For efficiency, NetFlow records are usually transported periodically via a UDP packet containing a 24 byte NetFlow header and multiple flow records, up to 30 (24 for v9) in a 1500 byte datagram. Consequently, records might be lost due to network error or congestion, particularly if experiment instrumentation support

Table 1: Partial NetFlow Record

Byte position	Contents	Description
0-3	srcaddr	Source IP address
4-7	dstaddr	Destination IP address
8-11	nexthop	IP address of next hop router
12-13	input	SNMP index of interface
14-15	output	SNMP index of interface
16-19	dPkts	Total packets
20-23	dOctets	Total number of L3 bytes
32-33	srcport	TCP/UDP source port
34-35	dstport	TCP/UDP destination port
37	tcp_flags	TCP flags seen
38	prot	IP protocol type
39	tos	IP type of service (ToS)

is poorly designed to support record generation rates.

Maintaining NetFlow data can be computationally expensive for a router (or software probe's host machine) and burden the host CPU or hardware to the point where it runs out of capacity. To avoid loading problems and reduce the volume of collection data, packets may be sampled; rather than examine every packet in a flow, sampled NetFlow records are estimates of the actual measured flow volume.

3.1 NetFlow in Emulab

Multi-tenant environment instrumentation presents a fundamental design challenge. The testbed operator requires system-wide monitoring to maintain overall system health, while experimenters require experiment-wide monitoring for performance validation and measurement. Testbeds such as Emulab emulate arbitrary network topologies using a flat L2 infrastructure and virtualization techniques such as VLANs. In some cases, switches such as the Cisco Catalyst 65xx-class used in *schooner* [6] provide native NetFlow support. However relatively few devices offered as client test resources currently support NetFlow natively. Though of potential benefit to experimenters, most testbeds don't export NetFlow data that can be obtained from infrastructure switches, nor offer mechanisms to interact (e.g., through NetFlow filters) with such data.

In Emulab LANs and links requiring traffic shaping operations such as packet loss or rate limits are emulated by compute nodes (i.e., *shaper* nodes). As we will see in Section 4 this presents additional complexity for automated monitoring deployments.

4 Goals, Design and Implementation

Instrumenting testbeds has received considerable attention in recent years [7], [8]. However testbed operators have first focused on offering *testbed-wide* rather than experiment-wide tools, and have made these tools visible to experimenters. One such example is PlanetLab's [9] *CoMon* [11], which tracks compute system performance and operation. CoMon data is archived, and experimenters may view either node-level or slice-level statistics to better understand experiment behavior.

The benefit of this approach for operators is that a tool of manageable complexity provides a service to both the operator and experimenters. But this approach provides little flexibility for experimenters who might seek, say, a finer grain view of their experiment. We envision the testbed operator providing tools that offer a targeted, controllable view of each individual active experiment. The goals we set for designing netflowize included:

- *Extensibility* Though our prototype deploys NetFlow probes and collectors, it is extensible and can be readily modified to create monitoring overlays intended to measure other network behaviors.
- *Flexibility* Experimenters need to be able to specify the granularity of monitoring data they collect. We expect the degree of granularity demanded by experimenters to evolve as an experiment progresses from a debugging phase to a final data collection phase.
- *Coverage* Though many tools focus on measurement at specific points in a topology, our emphasis was on building an experiment-wide data collection system.
- *Resource Usage Control* Monitoring demands compute and storage resources, and experimenters should be offered alternative overlay approaches that require use of varying quantities of resources, particularly in testbeds where node utilization is high.

As we will see in the following section, these design goals drove various design decisions.

4.1 Design

An experimenter must be able to balance instrumentation needs and available hardware support. Consider the network first. Hardware devices can easily generate tens of thousands of NetFlow records per second; a probe generating 10,000 NetFlow packets/second can consume 120

Mbits/sec of bandwidth. Since collectors aggregate traffic from many probes, their ingress links can be expected to reach capacity first. A collector running on a modern commodity server is roughly capable of processing 40K flows per second. We note that PlanetLab nodes may generate 100s of thousands of flows in a day. Today in practice we see much smaller numbers of active flows on Emulab. In our own project use, we rarely had occasion to exceed 500 simultaneous active flows on a 100 Mb link.

A key design problem we face with a netflow overlay is where to place probes and collectors in a arbitrary experiment topology, while seeking to strike a balance between 1) avoiding duplicate flow counting, 2) using the minimal required hardware support, and 3) obtaining complete experiment coverage.

4.2 Determining the Overlay Topology

Suppose we propose to modify an experiment to provide additional hardware and/or software resources for instrumentation (including probes and collectors). Determining the topology of such an overlay obviously demands knowledge of the experimental network.

A naive approach to overlay creation proceeds as follows; extract a network topology from its specification – the experiment's *ns* topology description – and then modify it to add required overlay hardware and software components. In principle, this approach can be executed prior to the experiment's instantiation. Yet consider the following example of a perfectly valid topology description:

Example 1:

```
$ns duplex-link [$ns node] [$ns node]\\
                          10Mb 0ms DropTail
```

Though perhaps bad form, Emulab will fill in unspecified details and create 2 nodes running the default operating system, and assign the nodes' names (e.g., *tbnode-n1* and *tbnode-n2*), and perhaps name the connecting link *tblink-l3*. Next consider a more common topology specification:

Example 2:

```
# create nodes
for { set i 0 } { $i < 2 } { incr i } {
   set node($i) [$ns node]
   tb-set-node-os $node($i) FBSD410-STD
}
# create link
set link0 [$ns duplex-link $node(0) $node(1)
                                10Mb 0ms DropTail]
```

Extracting node names is necessary to configure the collector that each probe instance is assigned to. In either above example, parsing the script to determine node naming is challenging; in some complicated scripts even determining the number of nodes requested is formidable.

The challenges of topology identification are further complicated by factors such as the insertion of nodes to implement link traffic shaping, which are invoked implicitly. While the need for shaping nodes can be determined from the script, multiple shapers can be implemented on a single shaper node; one can not easily determine the number of physical shaper nodes used in a given experiment instantiation. We note that other tools have circumvented the complexity of parsing and modifying scripts by choosing experimenter interaction. Deterlab's [10] Security Experimentation Environment (SEER) [13], [14], instructs the user to explicitly enter *ns* commands to create a control node, and install tarfiles and perform an initialization.

4.3 Post-Instantiation Modifications

The difficulties encountered in attempting to automatically determine a topology accurately and overlay an instrumentation infrastructure by modifying an *ns* file suggest that in many cases it is preferable to instantiate an experiment first – prior to its modification. The process proceeds as follows: instantiate an experiment, obtain the details associated with assigned resources, swap out the experiment, modify the ns-script (e.g., add nodes, deploy tarballs) and make other changes necessary for the instrumentation overlay, and swap the instrumented experiment back in.

The feasibility of such an approach relies upon 2 Emulab resource allocation system properties:

- *Persistence of resource assignments* Emulab resource assignments are 'sticky'. Resources assigned to an experiment are not immediately decommissioned and placed in the resource pool available for other experiments. Swapping an experiment out and back in will use the same resources. Persistent resources support powerful features such as an experimenter's ability to 'modify' an instantiated experiment.

- *Exposure of experiment resource instantiation details* Emulab exposes low-level details of experiment instantiation (e.g., switch ports) via an *XML-RPC* interface. Experimenters can get a list of all nodes, links, and associated names. The following example illustrates available low-level information for traffic shaping nodes.

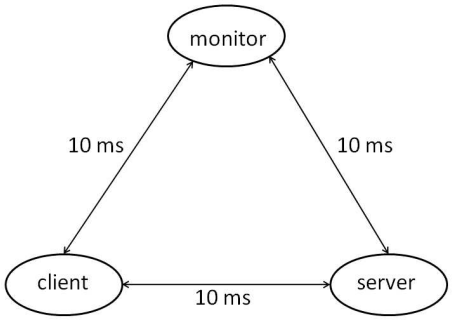

Figure 1: The number of 'shaper' nodes used to instantiate the 3 emulated links in this 3 node network can be difficult to predict.

Suppose we seek to implement the simple 3 node, fully connected mesh topology depicted in Figure 1. The number of physically distinct shaper nodes used to emulab the links can be difficult to know, and is determined in part by the number of physical interfaces available on shaper nodes. In this case two nodes are used to emulate the 3 links, and the *XML-RPC* invocation reveals node names, which can then be directly queried for the interfaces mapped to links between nodes:

```
tbdelay0: cat /var/emulab/boot/delay_mapping
link2 duplex client monitor fxp2 fxp3 60130 60140
link0 duplex client server fxp4 fxp1 60110 60120

tbdelay1: cat /var/emulab/boot/delay_mapping
link1 duplex monitor server fxp0 fxp1 60110 60120
```

Given link information gathered through *XML-RPC*, we are now prepared to construct a graph of our experimental topology. However our call only reveals details about links between compute nodes, and must still determine how shaping nodes fit in. As described above, we contact those nodes directly (e.g., via ssh) for attached link information, and construct the graph as follows:

1. Add all nodes to the graph (compute and shaping nodes).

2. Add links to graph starting with shaping nodes. Keep track of links to avoid redundant links.

3. Add links from compute nodes to graph.

4.4 Building the Overlay

The next step is to build an overlay. Experimenters should have considerable control over how the overlay is constructed. In particular, an experimenter should be able so specify whether an instrumentation overlay uses existing experiment resources, or whether new resources are to be requested.

Experimenters seeking to create the least intrusive or *heavyweight* overlay choose to incur the cost of acquiring additional nodes and links to run collector(s) and probes positioned as network taps. In doing so, experimenters ensure the least possible impact of measurement on their experiment's operation and performance, maximum experiment coverage, and the ability to generate the finest-grain measurement information. One case where a researcher might prefer this option is during the 'results collection' phase of their experiment's lifetime. Another example might be where a security researcher is interested in rapid detection of a malicious flow associated with a virus spread, and seeks frequent record updates sent to a collector. As an alternative, an experimenter might choose to collect as much netflow information as possible using spare capacity on existing compute resources. An experimenter might select this *lightweight* option during the 'debugging' phase of an experiment's lifetime.

Observations of our own experiments suggested that while compute node CPU usage was often high (see Figure 2), and the number of network interfaces on some nodes was exhausted, utilization of the Emulab control network infrastructure was mostly modest throughout experiment run-time. Hence, for either overlay construction mode we choose to conserve resources by always using the control network to serve as the measurement distribution network (i.e., transport network between probes and collector(s)) rather than create a dedicated measurement network. Of course, these starting points do not prohibit an experimenter from subsequently modifying the initial overlay construction to realize their desired measurement network infrastructure.

4.4.1 Lightweight Mode

- *Probe placement* To find the minimal number of probes needed to cover all network flows, we begin with an algorithm motivated by the *set cover* problem. We first consider each link's and LAN's attached nodes as a set, and start by picking the node that belongs to the most sets. On the selected node we run a probe responsible for monitoring the node's attached links. Repeat, but only on those sets

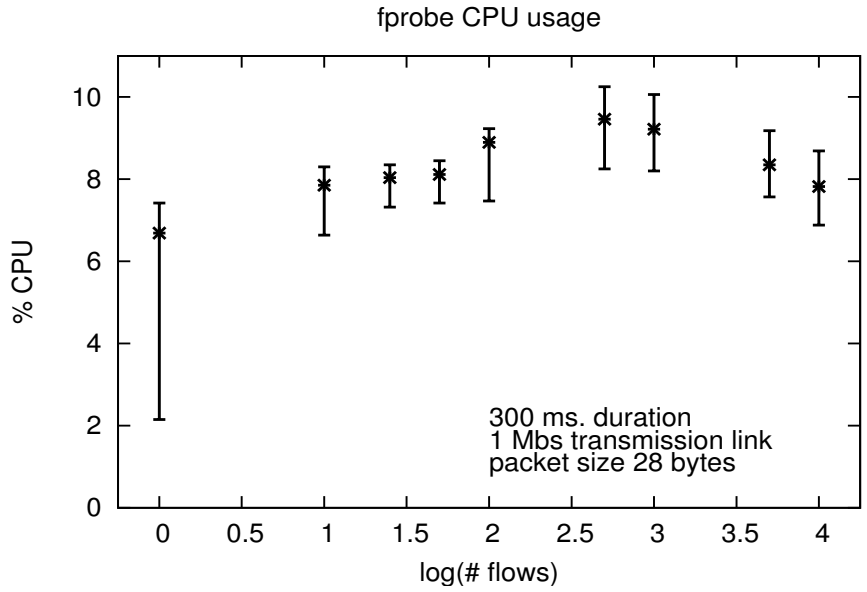

Figure 2: CPU utilization by probe on an Emulab PC850 node.

that have yet to be covered.

One complication is that a lossless LAN can be implemented on some systems with an actual switch (i.e., not a shaper bridge), and hence all traffic will not be forwarded to all attached links. Here the only approach ensuring traffic coverage is to run a probe on every attached node.

- *Collector placement* Though many optimizations are possible, we select a node at random, preferably from the set of nodes not operating as probes (if nonempty).

4.4.2 Heavyweight Mode

- *Probe placement* Each link in our experiment is replaced by a LAN connecting the attached endpoints, as well as a new node we attach to run a probe. A lossless LAN simply has an additional new node attached to run the associated probe. Lossy LANs continue to be probed from attached nodes.

- *Collector placement* Each collector is assigned to a newly assigned standalone node by default.

Before considering examples of tool operation in each mode, we note that there are several potential shortcomings with our current approach. First, it is unnecessary to force an experimenter to make a binary choice about resource consumption. One can imagine a more flexible approach would be to query a user for the number of additional resources that should be assigned for instrumentation purposes, and have the tool deploy exactly that number of resources to support probes or collectors in a most efficient manner. A related deficiency in our current approach to 'heavyweight' operational mode is that a user has no advance warning of the number of additional resources that will be required for the instrumented experiment. As a consequence, the experimenter can not be certain that sufficient resources are available in the system's resource pool.

5 Examples

A few simple examples demonstrate netflow overlay creation. Figure 3 depicts an experiment with 3 nodes interconnected with a LAN and no requested traffic shaping. End node shaping is explicitly enable to ensure that superfluous delay nodes are not instantiated. Exactly 3 compute nodes are assigned and interconnected with a switch. Netflowize's lightweight overlay deploys probes on 2 of the compute 3 nodes, recognizing that design's sufficiency to observe all flows. Minimizing network traffic, a single collector is also placed on a probe node.

In heavyweight mode on the same topology, netflowize modifies the *ns* script in a simple yet clever fashion. The single script modification is to disable end node shaping. When the experimented is swapped out and

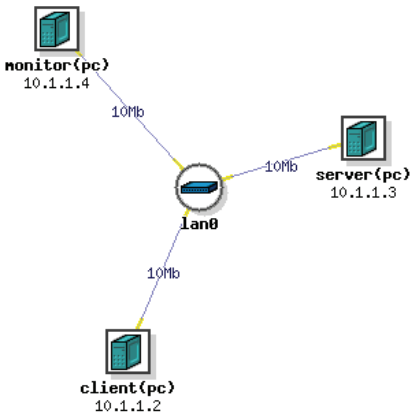

Figure 3: A simple 3 node topology.

back in (as necessary in heavyweight mode), two additional delay nodes are invoked to implement shaping (*tbdelay0* and *tbdelay1*), one between each of 2 end systems. NetFlow taps these two delay nodes to run probes to ensure full experiment coverage, and one of the delay nodes is also chosen as the collector.

5.1 Implementation

Netflowize comprises approximately 700 lines of Python code, and employs the widely used *flow-tools* open-source package for netflow record generation and collection. The tool is publicly available at http://66.92.233.103/netflowize-0.3.tar.bz2

6 Related Work

A wide range of distributed monitoring systems and approaches are closely related to our work, including:

- *PlanetFlow2* PlanetFlow [15] provides a testbed-wide view of flows on all PlanetLab node interfaces. PlanetLab operations relies on the data to investigate and resolve complaints received from third parties about suspicious or malicious traffic originating on a system node. A public web interface to the data set permits experimenters (and other researchers) to query either their own slice's or various aggregate node or system flow statistics.

  A custom version of *fprobe* is deployed on each PlanetLab node, the only major modification being support for collecting slice level flow information. The CPU overhead of the probe was reported to only be 1-3% under maximum load; PlanetFlow uses the substantially more efficient *fprobe-ulog* probe software. Under this configuration, it is reported that there is no need to resort to flow sampling. Rather than probes sending NetFlow records directly to a collector, they are stored on each node locally and periodically collected by a separate polling-based aggregation process. Real-time flow monitoring is of course prohibitive with this design, but it is sufficient for operational needs. The system as a whole is reported to generate up to 4 TB of data per day, the equivalent of roughly 390 Mbps of netflow packets.

- *Orbit Measurement Framework and Library (OML)*

  With its focus on experimenter control of instrumentation on the Orbit wireless testbed, OML [12] seeks to reduce the burden on experimenters of instrumenting their experiments. OML provides an API that permits individual experimenters to define measurement points and parameters, and collect and process measurement data.

- *SEER*

  SEER is an instrumentation workbench that enables DETER experimenters to simply conduct security experiments by providing agents for attack and traffic generation, collection and analysis. SEER integrates various tools for configuring and executing experiments and provides a user-friendly GUI for experimenters to use the tools. Like netflowize, SEER strives to make experiment instrumentation available to users at all skill levels. Netflowize could be viewed as a tool existing within a multi-purpose workbench like SEER.

- *Emulab Link/Node Tracing*

  By providing a native, testbed supported instrumentation tool that experimenters can flexibly employ on their experiment, Emulab's powerful *trace* feature is close in spirit to our approach. Trace permits packet capture and storage of packet traces at any link specified in an experiment's *ns* topology description.

- *Distributed Monitoring* The topic of probe placement for the fullest possible network coverage has

been widely studied [23], [24], [25]. Of particular interest, CSAMP [19] relies on feedback from netflow-capable routers on current traffic conditions to determine data collection points. Their goal was to reduce redundant flow collection and thereby increase flow coverage. Their work was focused from a network administrators standpoint where full topology, routing, and traffic matrix information was known.

- *DiMAPI* DiMAPI [20] creates an API to enable users to express complex distributed monitoring needs, choose only the amount of information they are interested in, and therefore balance the overhead with the granularity of information collected.

7 Discussion

We have described the development of *netflowize*, a new tool for researchers to analyze their testbed experiments. We believe that the tool will be particularly valuable for students and those unfamiliar with deploying and analyzing netflow. The tool might also be helpful for experiments that are exposed to the public internet to observe actual in-the-wild attacks. Security researchers using contained environments such as DETER will likely find the tool useful for application to anomalous flow detection, and perhaps tracking malware evolution [29], [30], [31], [32]. The tool might eventually also find benefit in traffic characterization in public multi-tenant clouds.

Many compelling topics were considered outside the scope of our project. This included crucial components of a complete system such as the visualization of gathered netflow data. We have also not directly addressed the specific monitoring needs that are related to the virtualization of end hosts and interfaces.

Some key lessons we learned included the benefits – and perhaps necessity – of experiment post-instantiation instrumentation. Just as crucial is the need for shared resource systems to expose even the lowest-level resource allocation and instantiation details to facilitate experiment monitoring.

References

[1] Cisco IOS NetFlow, http://www.cisco.com/en/us/products/ps6601/products_ios_protocol_group_home.html

[2] Emulab, http://www.emulab.net

[3] E. Pilli, R. C. Joshi, R. Niyogi, "A Generic Framework for Network Forensics," *International Journal of Computer Applications*, 1(11):1.6, 2010.

[4] Uddin, A., "Detecting Botnets Based on their Behaviors on Perceived from Netflow," http://courses.cs.ut.ee/2009/security-seminar/uploads/Main/mohammad-1.pdf

[5] Proceedings of *FlowCon 2011*, http://www.cert.net/flocon.

[6] Schooner, http://www.schooner.wail.wisc.edu

[7] Barford, Paul (Ed), "GENI Instrumentation and Measurement Systems (GIMS) Specification," *GENI Design Document 06-12*, Facility Architecture Working Group, 2007.

[8] "INSTOOLS: Instrumentation Tools for a GENI Prototype", http://groups.geni.net/geni/wiki/InstrumentationTools

[9] PlanetLab, http://www.planet-lab.org

[10] Deterlab, http://www.isi.deterlab.net

[11] CoMon, http://comon.cs.princeton.edu

[12] M. Singh, Ott, M, Seskar, I., Kamat, P., "ORBIT Measurements Framework and Library (OML): Motivations, Implementation and Features," *Proc. of TridentCom 2005*, 2005.

[13] Security Experimentation Environment (SEER), http://seer.deterlab.net

[14] S. Schwab, B. Wilson, C. Ko, A. Hussain, "A Security Experimentation Environment for DETER", *DETER Community Workshop on Cyber Security Experimentation and Test (CSET'07)*, 2007.

[15] PlanetFlow2, http://planetflow.planet-lab.org

[16] K. Sklower and A. Joseph, "Very Large Scale Cooperative Experiments in Emulab-Derived Systems," *DETER Community Workshop on Cyber Security Experimentation and Test 2007*, Boston, August 2007.

[17] K. Sklower and A. Joseph, J. Mirkovic, S. Wei, A. Hussain, B. Wilson, R. Thomas, S. Schwab, S. Fahmy, R. Chertov, and P. Reiher, "DDoS Benchmarks and Experimentation Workbench for the DETER Testbed", *Proc. of the 3rd IEEE Conf. on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom 2007)*, 2007

[18] Global Environment for Network Innovation, http://www.geni.net

[19] V. Sekar, M. Reiter, W. Willinger, H. Zhang, R. Kompella, D. Andersen, "CSAMP: A System for Network-Wide Flow Monitoring", *Proceedings of NSDI'08*, 2008.

[20] P. Trimintzios, M. Polychronakis, A. Papadogiannakis, M. Foukarakis, E. Markatos, Arne Oslebo, "DiMAPI: An Application Programming Interface for Distributed Network Monitoring", *Proc. of NOMS'06*, 2006

[21] *Flow-tools*, http://www.splintered.net/sw/flow-tools /docs/flow-tools.html

[22] System for internet-Level Knowledge (SiLK), http://tools.netsa.cert.org/silk/

[23] Cantieni, G. R., Iannaccone, G., Barakat, C., Diot, C., Thiran, P., "Reformulating the Monitor Placement Problem: Optimal Network-Wide Sampling", *Proc. of ACM CoNeXT 2006*, 2006

[24] Chadet, C., Fleury, E., Lassous, I., Herve , Voge, M.-E., "Optimal Positioning of Active and Passive Monitoring Devices," *Proc. of CoNeXT 2005*, 2005

[25] Suh, K., Guo, Y., Kurose, J., Towsley, D., "Locating Network Monitors: Complexity, heuristics and coverage", *Proc. of IEEE INFOCOM 2005*, 2005

[26] Estan, C., Keys, K., Moore, D., Varghese, G., "Building a Better NetFlow", *Proc. of ACM SIGCOMM'04*, 2004

[27] Kompella, R., Estan, C., "The Power of Slicing in Internet Flow Measurement", *Proc. of IMC 2005*, 2005

[28] Sharma, M., Byers, J., "Scalable Coordination Techniques for Distributed Network Monitoring", *Proc. of PAM 2005*, 2005

[29] Lakhina, A., Crovella, M., Diot, C., "Diagnosing Network-Wide Traffic Anomalies", *Proc. of ACM SIGCOMM'04*, 2004

[30] Mai, J., Chuah, C.-N., Sridharan, A., Ye, T., Zang, H., "Is Sampled Data Sufficient for Anomaly Detection?", *Proc. of IMC 2006*, 2006

[31] Sekar, V., Duffield, N., Van Der Merwe, K., Spatscheck, O., Zhang, H., "LADS: Large-scale Automated DDoS Detection System", Proc. of USENIX ATC 2006, 2006

[32] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A., "An Integrated Experimental Environment for Distributed Systems and Networks", *Proc. of OSDI'02*, 2002