Router Primitives for Programmable Active Measurement

Joel Sommers
Colgate University
jsommers@colgate.edu

Paul Barford
University of Wisconsin-Madison
Nemean Networks
pb@cs.wisc.edu

Mark Crovella
Boston University
crovella@cs.bu.edu

ABSTRACT

Active probe-based measurements are the foundation for understanding important network path properties such as SLA compliance and available bandwidth. Well-known challenges in active probe-based measurement include the logistics of deploying and managing host-based measurement infrastructures, the load that probe packets place on network resources, the inaccuracy of resultant measurements, and the relatively limited set of features that can be measured. In this paper, we argue that these challenges can be addressed through programmable, router-based support for active measurement. While commercial routers today have some basic capabilities for emitting probe packets, these mechanisms are minimal and do not allow the necessary flexibility in the kinds of probing that can be done. We describe a set of functional primitives that enable a wide range of router-based active measurements and would improve and simplify the ability to assess and understand network structure and dynamic network state. We discuss the associated resource requirements and implications of our approach related to configuration, security and privacy. Finally, we support and illustrate the powerful potential of our approach through a series of measurement scenarios and describe our ongoing efforts toward a Click-based implementation of our framework.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network management, Network monitoring; C.2.5 [**Local and Wide-Area Networks**]: Internet (*e.g.*, TCP/IP); C.2.6 [**Internetworking**]: Routers; C.4 [**Performance of Systems**]: Measurement Techniques

General Terms

Design, Experimentation, Measurement

Keywords

Active Measurement, Programmable Measurement, Router Programmability

PRESTO'09, August 21, 2009, Barcelona, Spain.

1. INTRODUCTION

The ability to measure and assess network path characteristics such as SLA compliance, available bandwidth and the underlying topological structure is critical in the design and development of network systems and protocols and in day-to-day network operations. The primary means for gathering data on network path characteristics is through the use of active probe-based techniques that focus on generating carefully crafted sequences of packets and observing characteristics of those packets after they have traversed a path.

There are two classes of challenges in active probe-based measurement: technical and logistical. Among the technical challenges are issues related to precision in the generation and reception of packet probe streams, including accurate timestamping. A key logistical challenge is that there is essentially no intrinsic support for active probe-based measurements in the Internet. Thus, deployment and management of additional infrastructure specifically for active measurement is generally required. Furthermore, it is well known that some standard techniques such as traceroute-like probing are often blocked by service providers who are concerned about the load that these measurements place on their routers.

In this position paper, we argue that routers are in a unique position to provide programmatic support for active probe-based measurement of network paths. If available, this capability could provide a number of important advantages. First, router-based active measurement would mean that no additional infrastructure would be required to measure path properties. Second, by virtue of their in-line perspective, probing methods could be designed to virtually eliminate concerns over accuracy and the interference of probing on customer traffic. Third, programmability would mean that entirely new probing techniques could be developed to enable a broader range of path characteristics to be measured. Fourth, when coupled with carefully considered access control policies, router-based measurement could open the door to a greater overall understanding of the Internet's structure and behavior. Finally, we note that our proposal for implementing a programmable probing capability "deeper" in the network in order to improve performance and accuracy is motivated by a key aspect of the end-to-end arguments [13].

In this paper, we describe an architecture and prototype implementation framework for programmable active measurement capability in routers. The high level requirements of the architecture include: (1) flexible and accurate probe generation and reception; (2) the ability to measure nodes (*e.g.,* router delay), links (delay between two nodes) and/or paths (delays between multiple nodes); (3) flexible specification of measurement experiments (paths taken by probes, targets of probes, experiment start and duration); (4) low space and computational overhead; and (5) flexible access control.

Our implementation framework is designed to meet the above

requirements and consists of a set of functional primitives designed around the concepts of *events* and *actions*. An example of an event is the expiration of a timer. Events trigger actions such as the generation of a probe packet. We describe our set of primitives and highlight their utility through a series of illustrative examples. While our initial set of primitives provides a great deal of intrinsic probe-based measurement capability, it remains a work in progress. Our study is in its early stages and we are presently developing an implementation in Click [6] in order to better assess the details of hardware impact and the design and scope of our set of primitives. We remain confident that our ongoing implementation work will underscore the transformative potential of router-based programmable measurement.

2. RELATED WORK

In-network active measurement is valuable and ISPs already do some of it in an *ad hoc* way. For example, many ISPs set up servers to allow in-network generation of traceroute probes. Further, a recent effort [1] has set up in-network tools to aid users in understanding the performance of their providers. These capabilities could be provided more completely and simply through the deployment of the methods we propose in this paper.

In commercial routers today, there exist various capabilities to facilitate both passive and active network measurements. While a range of configuration options are available for these facilities, there are no *programmable* capabilities. For active measurements in particular, there are facilities to generate probes according to preset emission processes, perform basic statistics on these measurements, and generate SNMP traps when statistics cross a configured threshold (*e.g.*, if delay exceeds a given value) [2, 3]. While our focus is also on router capabilities, we propose a set of primitives that can be composed by a user to achieve a much wider variety of measurement objectives.

Extensive facilities for router-based measurement were a feature in the early ARPAnet Interface Message Processors. According to Kleinrock and Naylor, "...we were careful to include in every specification of the network design the ability to monitor network behavior with the use of specific measurement tools" [5]. For example, the IMPs had a capability to record four timestamps over the course of processing a given packet in order to examine detailed performance characteristics. There were also capabilities to accumulate various statistics (*e.g.*, histograms of packet sizes), the ability to take a snapshot of system resources and an "artificial message generation" capability, *i.e.*, the ability to send probe packets. The authors point out that the ARPAnet had two goals: (1) experimentation with packet switching, and (2) providing connectivity. It appears that by 1974 goal (2) had pushed out goal (1) and these measurement capabilities were dropped. Some of the capabilities we propose are similar to those offered in these early packet processing systems.

There have recently been a number of proposals for programmable *passive* measurement systems. Ramaswamy *et al.* [12] describe a programmable passive measurement system based on network processor-based systems deployed in a network. More recently, Yuan *et al.* describe the ProgME system which is based around the notion of capturing *flowsets* (arbitrary collection of flows) [17]. In [4], Khan *et al.* examine an FPGA-based *query-driven* collection engine that can be reprogrammed based on user/application requirements. Our focus is different in that we propose a set of capabilities for router-based active measurement rather than on capabilities of an external passive measurement system.

Most closely related to our work is that by Sommers and Barford in which they describe a flexible system for facilitating active measurement in a shared testbed such as Planetlab [14]. However, their focus is on end-hosts and the range of capabilities offered is quite narrow. In a somewhat similar vein, the Scriptroute system provides special library support in the Ruby language to facilitate active measurements [16] in an environment like Planetlab. More loosely related to our efforts, Machiraju and Veitch propose flexible priority handling of probe packets at routers in order to isolate performance measures at different hops a long a path [9].

Finally, The IP Measurement Protocol described by Luckie *et al.* [8, 7] is related to our effort in the sense that IPMP is designed to facilitate router-based measurement. The payload of an IPMP packet consists of path records (*e.g.*, timestamps, interface IP addresses) that can be added by routers as the packets are forwarded. Similarly, Pezaros *et al.* discuss extensions to a IPv6-based "active network" software router designed to augment packets with passively collected measurement data (*e.g.*, to add timestamps to packets as they are forwarded through the software router) [11].

3. SYSTEM DESIGN

In this section we first discuss our general objectives for a set of primitives for enabling programmable router-based active measurement. We then describe our proposed primitives, their function and purpose. Next, we discuss resource requirements of these primitives. Finally, we present examples of the kinds of measurements possible with our design.

3.1 Objectives

The objectives behind the design of the set of primitives described below are as follows:

1. **Permit flexibility in specifying the probe emission and processing by intermediate routers.** Our first objective is to embed as few assumptions as possible in the active measurement mechanisms available in a router. The available primitives should enable a wide range of probing processes and algorithms to be implemented, including stream-based and non-stream-based probing. Users should be able to specify probe packet sizes, payloads, spacing among packets, and so forth.

   In addition to flexibility in probe packet emission, the available primitives should provide latitude in the kinds of processing that can be done upon probes at intermediate routers along a path. For example, a user should be able to control or influence the route that probes take from a source to a destination. Users should be able to specify how a probe packet might be modified in order to include information as the probe is handled by a given router, *e.g.*, to add a high-resolution timestamp or to add an interface address.

2. **Improve the accuracy of active measurement.** The router primitives should enable improvements in the accuracy of active measurements, as compared to what is possible using current (end-system) active measurement approaches. Routers are in a unique position to be able to provide information that can yield insight into network performance and behavior.

3. **Optionally suppress probing-induced effects.** Active measurements impose extraneous load on networks that can cause unwanted packet delays or packet loss. An explicit objective of our design is to be able to avoid these effects where reasonably possible. Clearly, this should only be optional; *e.g.*, a probe intended to measure packet loss would not use it.

However, for some kinds of probing (*e.g.*, probing for topology information), a router is in the position to handle probes in such as way as to avoid affecting non-probe traffic.

4. **Permit multiple, simultaneous users.** Multiple users should be able to use the primitives available in routers simultaneously. For example, multiple operations groups within an ISP may wish to initiate probings in order to measure a characteristic of interest; they should be able to do so at the same time.

5. **Provide secure access for users along with selective user rights.** The router primitives should not be accessible to users without proper credentials. Furthermore, an ISP may wish to allow some users access to certain primitives but not others; the ISP should have the flexibility to assign different rights to users. For example, the network operations center for a friendly peer ISP may be given access to certain primitives in order for that peer to troubleshoot customer problems. However, use of all primitives might be disallowed so that the peer cannot gain "too much" information.

6. **Provide the ability to enforce limits on resource usage.** In addition to securing access and providing selective access to the router measurement primitives, accounting is an important capability that should be provided. Probe traffic quotas should be supported as well as processing time quotas. For example, it may be desirable to set an upper limit on traffic induced by a particular user over a given time interval between two routers or across an entire network. Practically, it may be very difficult to support *hard* limits, *i.e.*, ensuring that no limits are ever exceeded, across an entire network. On the other hand, *soft* limits, in which a user may temporarily exceed a threshold, may be possible and still provide the key benefit of rapidly detecting user errors or deliberate misuse.

7. **Low impact on router.** The measurement primitives should impose as little computational and memory load requirements on a router as possible. Most importantly, a programmable active measurement capability should have negligible performance impact on the standard processing of packets on the data plane (and no negative performance impact on other router subsystems).

3.2 The Primitives

In our system design, there are two types of primitives: *events* and *actions*. Simply put, particular actions or sequences of actions can be tied to the occurrence of an event. Below, we describe the events of interest in our current design, and the various actions that can be performed. For each primitive, we introduce simple assembly-like instructions and their semantics. In Section 3.4 we present a set of scenarios to show how the primitives can be used together to achieve different measurement goals.

Note that in this work, we assume that we are concerned only with measuring data plane activity. Thus, measurements or events related to the control plane are out of scope.

3.2.1 Events

The occurrence of particular events can trigger the execution of actions. In our current design, there are two events of interest:

- **Timer expiry.** An action or sequence of actions can be associated with the expiration of a timer that has been previously set. Timers may be set with the `after` action primitive, described below.

- **Probe packet receipt.** An action or sequence of actions can be associated with the arrival of a packet that matches certain criteria. In this work, we do not specify the syntax or scope of the kinds of criteria that may be used to match incoming packets. At minimum, the ability to match IP source and destination addresses, protocol number, and source and destination ports should be sufficient in order to identify incoming probe packets. These matching capabilities are standard features in router access control lists; we propose that these basic ACL capabilities be augmented to allow a set of measurement primitive actions to be performed upon a successful match. Further, to meet our objective of supporting multiple simultaneous users, probes should be distinguishable on a per-user basis. This will require a unique identifier shared between each probe packet and its associated action code in each router.

3.2.2 Actions

When an event has fired, a sequence of one or more actions may be performed to process an incoming probe packet or to generate new probe packets. The action primitives in our system design are as follows:

- **Set a timer.** The `after` primitive can be used to set a timer so that a sequence of actions is performed in the future. This primitive takes two parameters: the amount of time after which the timer should expire, and an identifier indicating the action of a sequence at which to start execution on timer expiry. When the `after` action is encountered, execution is halted until the timer expires, at which time execution begins at the identifier given in the statement. Thus, the following set of actions causes a timer to be fired every second at which time nothing is done except to reset the timer.

```
start:
   after 1.0 start
```

- **Forward a packet.** The `forward` action may be used to control the path that a probe packet takes through a network. The `forward` action takes one parameter, the *destination*. The destination may be a specific IP address, or it may be the keyword `next-hop`, indicating that the packet should be forwarded to the next hop towards the destination in the probe's IP header, according to the router forwarding table.

  Note that the `forward` action enables a user to send a packet out an interface not on any standard forwarding path, thus creating the capability to probe over an alternative IP routing configuration. One example application of this facility could be to test router ACLs by forwarding probe packets to a router that should drop them, and testing whether the packets are indeed dropped.

- **Create and send a packet.** The `probe` action can be used to generate and send a new probe packet. Parameters to the `probe` action specify the packet header details for the probe to be generated.

  `probe` *destination* [*optional parameters*]

  Optional parameters might be used to specify the source address, destination address, protocol, port numbers, payload, outgoing interface or next-hop, etc.

- **Append a timestamp to a packet.** A key action to perform at a router is to append a high-resolution timestamp to a probe packet. This action can be performed immediately on

packet receipt using the `input-timestamp` action, and/or when it is queued to be forwarded to the next hop with the `output-timestamp` action.

In this work, we assume that a router has a high-resolution timestamp capability in order to provide these actions. Furthermore, we do not specify *how* the timestamp is appended to the probe packet. One possibility would be to use the idea from the IP Measurement Protocol proposal of *path records* to facilitate adding this information [8].

- **Append an interface address to a packet.** Similar to the addition of timestamps, addresses of the input and/or output interfaces may be appended to a probe packet with the actions `input-address` and `output-address`.

- **Append passive measurement data to a packet.** More generally, data that is available through passive means at a router may be appended to a probe packet, *i.e.*, information available through SNMP. For example, we might request that current output queue size be appended to a probe packet, or we might request that the input packet count be appended. The `input-mib` and `output-mib` actions can be used to collect such `input-address` and `output-address` information. One parameter is given to these actions, the MIB object identifier Note that any MIB table index (*i.e.*, which interface) is implied by the arrival or departure interface of the packet.

  For example, to collect the `ifHCInOctets` counter (from the Interfaces Group MIB [10]) from an input interface, as well as a timestamp, the following would suffice:

```
input-timestamp
input-mib 1.3.6.1.2.1.31.1.1.1.6.
```

- **Store a packet for subsequent retrieval.**

  If a router is the destination of probes, the probes can be stored for later retrieval by a user using the `store` action.

To the above actions, two conditional clauses may be used: `if` and `when`. With these conditional clauses, an action can be performed if a condition is true, or an action can be performed when a condition becomes true, *i.e.*, wait until a condition is true, then perform the action.

At present, we allow the standard binary comparison operators $==, !=, <, >, <=$ and $>=$. We allow numeric operands as well as three keyword operands: `outputqueue`, `empty`, and `full`. Thus, to delay forwarding an incoming packet while the output queue is non-empty, the following expression could be used:

```
forward next-hop when outputqueue == empty
```

Finally, a definite loop statement `repeat` is provided as part of the action primitive set. The `repeat` action takes two parameters: an iterator variable and the number of iterations to perform. The iterator variable takes on the values $0..n-1$, where n is the number of iterations specified. The set of actions to be repeated is delimited by the `repeat` and subsequent `endrepeat` keywords. Thus, the following statements cause three probes to be emitted back-to-back. The payload for each packet is a sequence number (0..2) as a four-byte integer.

```
repeat i in 3:
  probe 10.0.1.1 udp dport 3000 payload {i/4B}
endrepeat
```

3.2.3 Execution Model

When and how are the actions executed? While the answer to this question likely depends on the router hardware architecture, our initial design and implementation is conceived in the context of a shared memory router with a central processor. We believe that our system will be able to be naturally extended to distributed router architectures.

For each event that is fired, a virtual thread of execution is spawned to process the sequence of actions. The set of action primitives has been designed for safety with respect to memory and CPU resource usage. It precludes arbitrary-length or infinite loops and does not allow access to arbitrary memory locations. The goal is that the resource demand (CPU and memory) of any action set be statically analyzable (*i.e.*, offline); this allows the use of an admission control procedure for accepting requests to install measurement code. A small set of numeric variables are preserved across invocations of virtual threads, similar to the MAD system [14]. These variables can enable a user to maintain sequence numbers and other information over time. To manipulate these variables, a basic set of arithmetic instructions are provided, as well as instructions for generating random numbers, again similar to the instructions available in [14].

3.2.4 System Coordination

Although not explicitly part of our design, an external host is necessary for requesting installation or removal of events and actions in different routers of interest. It is this host that would translate a high-level measurement plan into primitives to be set up at routers in the network. For example, a general plan such as "measure router processing delays at each hop along a path" might be translated into a sequence of primitives to be installed in the routers of interest. It is also this host that completes the control loop for any reactive-type probe algorithms, *e.g.*, a stream-based available bandwidth measurement methodology in which the probing may change over time based on results of past probes. An external host is also necessary for retrieving stored probes and for any additional processing of the probes. (Below, we describe how this host would gain access to the system and present authentication and authorization information to a router.) At this time, we do not specify the control protocol(s) by which an external host interacts with a router or with the authentication system described below.

3.3 Resource Requirements

Given the simplicity of the primitives and restrictions placed on the number of variables that persist across executions of action sequences, we expect the computational and memory overheads of our system to be small. Existing access control list mechanisms can be extended to support packet arrival events, and there are also timers and scheduling mechanisms that can likely be extended to support timer events.

Some amount of memory will need to be reserved for packets held due to `when` clauses; a conservative maximum can be placed on this amount of memory in order to bound this cost. A control interface to query and/or flush packets held due to `when` clauses would provide the ability to test whether any probes are being held pending a condition becoming true. Furthermore, we note also that memory costs related to the `store` action would likely be far less than is used for current passive flow measurement capabilities in routers.

Depending on traffic conditions, there may be situations in which packets waiting on a `when` condition consume memory resources for an extended period of time. In such situations, there may not be sufficient storage for an arriving packet that requires `when` process-

ing. How should these conditions be handled? Generally speaking, the measurement activity responsible for the probes should be quickly notified, and may need to be paused or stopped altogether while the situation is resolved. We have not yet defined the mechanisms for performing such error notification and system-wide exception processing.

Aggregate resource costs at a router for all users of the measurement primitives can be bounded through accounting mechanisms and policies that allow only a certain number of users to access the measurement primitives (and which primitives) at a given time. Below, we describe how such policies could be implemented.

3.4 Illustrative Examples

We now describe a set of examples that illustrate the powerful potential of our set of measurement primitives.

Existing standard end-to-end probing algorithms. We first note that well-known existing probing mechanisms are easily implemented using our primitives. For example, the following timer-based action sequence could be used to implement the Badabing packet loss probing algorithm [15]:

```
set seq 0
set slot 0
nextprobe:
  repeat i in 3:
    probe 10.0.1.1 udp dport 3000 \
      payload {slot/4B seq/4B i/4B}
  endrepeat
  ; if this is the first probe of a pair
  if slot % 2 == 0:
    ; send next probe at next time slot
    set next 1
  else:
    ; otherwise, wait for some
    ; geometrically distributed number of
    ; time slots before sending next
    set next geom-rv
  slot += next
  seq += 1
  ; wait 'next' discrete time intervals
  ; before sending next probe
  wait = next * 0.005
  after wait nextprobe
```

In the above code, the `slot` variable holds the discrete time slot at which a probe triple is sent. If a slot is even, the next probe is sent at the next time slot, otherwise it is sent at a geometrically distributed random interval. The duration of each time slot is 5 milliseconds.

Minimal-impact record route. A record route implementation that applies both input and output interface addresses to a probe packet is given below:

```
input-timestamp
input-address
output-address
forward next-hop when outputqueue == 0
output-timestamp
```

This action sequence could be executed given a matching incoming packet. Perhaps most importantly, this record-route implementation is designed not to cause any packet loss at the output queue through use of the `when` clause. Such an implementation minimizes impact on network links and customer traffic, though clearly there is some impact on the routers as they must process the measurement primitives.

Router-internal delay measurement. Through simple modification of the above action sequence, delay measurement internal to a router can be captured. As shown below, if the `when` clause is removed, we see that two timestamps are added to a probe packet, one on input, and the other at output. The difference in these timestamps could give a measure close to the amount of time a packet takes to pass through a router. Again, this action sequence could be triggered through the arrival of a particular matching packet.

```
input-timestamp
input-address
output-address
output-timestamp
```

"Drive-by" passive measurement collection. Using the actions that enable collection of MIB data, a probe packet could collect a set of related data as it traverses a path through the network. For example, routers along a path might be configured to append the input and output byte counters to a probe. By sending multiple probes over time, an estimate of spare capacity along a path could be collected. To accomplish this, the following code segment might be installed in the routers of interest:

```
; include interface time stamp
; and octet count on ingress
input-timestamp
input-mib 1.3.6.1.2.1.31.1.1.1.6.

; include interface time stamp
; and octet count on egress
output-timestamp
output-mib 1.3.6.1.2.1.31.1.1.1.6.
```

Moreover, consider a scenario in which not only are specific MIB data appended to a probe, but the probe forwarding path is constructed (using the `forward` action) such that the probe visits all routers in a network, or routers within a particular region of a network. Such a capability would enable an efficient collection of passive measurement data.

The capability to collect drive-by passive data opens up the possibility to collect this information in conjunction with active probing as a way to compare and calibrate active and passive measurements. For example, the final packet of a probe stream designed for measurement of available bandwidth could be augmented with sufficient information to compute a passive measure of available bandwidth. At present, such calibration measurements typically require external high-precision passive measurement equipment.

4. SECURITY AND ACCESS CONTROL

In order to gain access to a router for adding or removing an event trigger and associated actions, a user must obtain a *ticket* from a central authentication and authorization system. The ticket has a limited lifetime and is encrypted using a secret key that is shared between the auth system and given router, *i.e.*, the user cannot modify the ticket.

The ticket includes timestamp information (*i.e.*, when the ticket expires), a unique user identifier, the source address for the user, a unique identifier for the router, a *static capability* set and a *dynamic capability* set. The unique user identifier can be used in the probe packet to allow demultiplexing of probe packets on arrival at a router and to match an incoming probe to its associated event actions. The static capability identifies the action and event primitives that the user is allowed access to for the router. For example, a user might not be given access to actions that enable gathering of MIB data, or to specify an address other than `next-hop` for the `forward` action. The dynamic capability contains information regarding quotas and other limits that must be monitored over time for adherence. For example, a user might be limited to some maximum bandwidth for probe traffic. Due to space constraints, we omit

discussion of issues related to capability revocation in the case of, *e.g.*, exceeding a bandwidth limit.

A user must obtain separate tickets for each router in which the user wishes to install events and actions. Because a ticket contains the user's source address and a router identifier, it cannot be transferred to another user or used with a different router. The central broker must keep track of a user's currently active tickets in order to possibly issue any capability revocations or to deny issuance of a ticket in the case that a user already has exceeded some threshold of outstanding tickets. Each router with events and actions installed for a given user must maintain accounting information such as processing and memory usage in order to monitor dynamic capabilities and also to monitor system-wide usage of the measurement primitives. If, in processing user actions, a dynamic capability threshold is exceeded, the router informs the central broker which issues ticket revocations to any affected routers. Events and actions for the user are then temporarily disabled. User access may be automatically re-enabled when the ticket expires, or upon manual intervention by an administrator. Note that each router must maintain a ticket revocation list so that any future attempts to use the ticket (within its lifetime) are disallowed.

Fine-grained capabilities allow the possibility to create, for example, policies that allow a friendly peer ISP limited access for collecting certain types of measurements in order to troubleshoot problems. VPN customers could also be granted limited access in order to measure performance across a provider's network and to verify compliance with SLAs. Events and actions could be installed for anonymous users in order to enable some types of measurements by outsiders. For example, non-authenticated users could be given limited access for collecting routing information. If too many probes were observed, the dynamic capability processing could cause the access to be temporarily disabled.

5. SUMMARY AND CONCLUSIONS

In this position paper, we argue for programmable active measurement capability in routers. Routers are in a distinctive position for supporting accurate and flexible active measurement, and router-based measurement programmability could enable new and fundamentally transformative ways of understanding network performance and behavior. Our framework for router-based programmable active measurement consists of two types of primitives: *events* and *actions*. We describe each of our proposed primitives and examined the potential of our framework through a set of measurement scenarios.

Our ongoing work is to develop an implementation of our framework in the Click modular router. As we gain experience with using our proposed primitives, we expect to refine them and to develop a better understanding of their potential memory and computational cost. We also expect to understand better the range of measurement methodologies enabled through our framework.

Acknowledgments

We thank the anonymous reviewers for their constructive comments. This work was supported in part by National Science Foundation (NSF) grants CNS-0347252, CNS-0627102, and CCR-0325701. Any opinions, findings, conclusions or other recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the NSF.

6. REFERENCES

[1] Measurement lab. `http://www.measurementlab.net/`.

[2] Cisco IOS IP SLAs. `http://www.cisco.com/go/ipsla`, 2009.

[3] JUNOS Real-time Performance Monitoring Services Overview. http://www.juniper.net/techpubs/software/junos/junos74/-swconfig74-services/html/rpm-overview.html, 2009.

[4] F. Khan, L. Yuan, C.-N. Chuah, and S. Ghiasi. A programmable architecture for scalable and real-time network traffic measurements. In *ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, November 2008.

[5] L. Kleinrock and W. Naylor. On the measured behavior of the ARPA network. In *AFIPS Conference Proceedings, National Computer Conference*, volume 43, May 1974.

[6] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3), August 2000.

[7] M. Luckie and T. McGregor. Path diagnosis with IPMP. In *NetT '04: Proceedings of the ACM SIGCOMM workshop on Network troubleshooting*, 2004.

[8] M. Luckie, T. McGregor, and H.-W. Braun. Towards improving packet probing techniques. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 2001.

[9] S. Machiraju and D. Veitch. A measurement-friendly network (MFN) architecture. In *INM '06: Proceedings of the 2006 SIGCOMM workshop on Internet network management*, 2006.

[10] K. McCloghrie and F. Kastenholz. The Interfaces Group MIB. IETF RFC 2863, 2000.

[11] D. Pezaros, M. Sifalakis, M. Schmid, and D. Hutchison. Dynamic link measurements using active components. In *Sixth International Working Conference on Active Networking (IWAN'04)*, October 2004.

[12] R. Ramaswamy, N. Weng, and T. Wolf. A network processor-based passive measurement node. In *PAM Workshop*, 2005.

[13] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4), November 1984.

[14] J. Sommers and P. Barford. An active measurement system for shared environments. In *Proceedings of ACM Internet Measurement Conference*, October 2007.

[15] J. Sommers, P. Barford, N. Duffield, and A. Ron. Improving Accuracy in End-to-end Packet Loss Measurement. In *Proceedings of ACM SIGCOMM*, August 2005.

[16] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A Public Internet Measurement Facility . In *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[17] L. Yuan, C.-N. Chuah, and P. Mohapatra. ProgME: towards programmable network measurement. In *ACM SIGCOMM*, August 2007.