SDN as Active Measurement Infrastructure

Erik Rye*, Robert Beverly†

*US Naval Academy
†Naval Postgraduate School

March 1, 2017

Active Internet Measurements (AIMS) Workshop
Motivation

Active Measurement Infrastructure

Today:
- Requires dedicated measurement nodes (e.g., Pi’s, end-hosts)
- No standard interface or API
- Limited extensibility
- Hard to deploy

Our vision:
- Active measurement integrated into existing routers and switches
- Standards-based API for probing and receiving results
- Quickly create and deploy new measurement tasks
- Measure from the network core – rather than edge
Motivation

Active Measurement Infrastructure

Today:
- Requires dedicated measurement nodes (e.g., Pi’s, end-hosts)
- No standard interface or API
- Limited extensibility
- Hard to deploy

Our vision:
- Active measurement integrated into existing routers and switches
- Standards-based API for probing and receiving results
- Quickly create and deploy new measurement tasks
- Measure from the network core – rather than edge
Motivation

Our Vision

SDN as Active Measurement Infrastructure (SAAMI):
- Leverage Software Defined Networks (SDNs) for active Internet measurement

SDNs:
- Commodity network forwarding hardware programmed via centralized controller
- Widely deployed / supported in hardware and software
- How to use for active measurement?
SDN as Active Measurement Infrastructure (SAAMI):

- Leverage Software Defined Networks (SDNs) for active Internet measurement

SDNs:

- Commodity network forwarding hardware programmed via centralized controller
- Widely deployed / supported in hardware and software
- How to use for active measurement?
Intuition: SDNs provide the basic building blocks for programmable active measurement:

- Controllers construct arbitrary packets, instruct switches to emit them out specified port
- Install packet match rules in switches to redirect measurement responses to controller
- Controller can perform arbitrarily complex computation over received measurement responses
Motivation:

1. **Lowers VP deployment barrier:** Utilize large existing deployed base of SDN infrastructure. Place measurements anywhere an SDN switch exists without installation, maintenance, or policy hurdles.

2. **Lowers VP diversity barrier:** Place VPs in the network core without consuming an interface or valuable space / power.

3. **Lowers VP utilization barrier:** Standardized OpenFlow permits rapid creation and deployment new measurement tasks and protocols.
Proof-of-concept: ping, traceroute

- SAAMI controller provides a RESTful API for ping
- Controller calibrates timing via OFEcho
- Emits ping probe via OFPacketOut
- Responses shunted to controller via OFPacketIn

Q: What’s the real-world feasibility?
SAAMI controller provides a RESTful API for ping
Controller calibrates timing via OFEcho*
Emits ping probe via OFPacketOut
Responses shunted to controller via OFPacketIn

Q: What’s the real-world feasibility?
Large-scale testing

- Probe 15,000 IPv4 targets
- From both OpenVSwitch (OVS) and hardware HP2920
- Using both local and remote SAAMI controller
SAAMI facilitates *new* functionality:

- Consider classic router aliasing and ownership inference problems
- Imagine provider wishes/compelled to add “routerID” functionality to her network for management and debugging
- Define ICMP type 200 code 0 packets as “routerID” query
- Using SAAMI, create a switch rule and respond with device’s AS and a unique identifier
Custom Measurements

**Router ID:**

- While any database could provide identical functionality, SAAMI closely couples measurement (which knows AS and router identifier) to control plane.
- Only a few lines of code – demonstrates the ease with which new measurement protocols can be deployed operationally.
- Provides functionality not possible in today’s hardware. While a simple example, it effectively solves aliasing and ownership problems.
Really simple routerID implementation!

```python
icmp = dpkt.icmp.ICMP()
icmp.type = 200
icmp.code = 0
icmp.data = 'router_id_query'

s.connect((sys.argv[1], 1))
print(('str(icmp)'))
```

Really simple SAAMI routerID response!

```python
p = packet.Packet()
e = ethernet.ethernet(dst=self.gwMAC, src=self.ownMAC, ethertype=ether_types.ETH_TYPE_IP)
i = ipv4.ipv4(src=self.ownIP, dst=ip.src, proto=1)
probe = icmp.icmp(type_=200, code=1, data=ROUTER_ID)
p.add_protocol(e)
p.add_protocol(i)
p.add_protocol(probe)
p.serialize()
actions = [parser.OFPActionOutput(self.gwPort)]
out = parser.OFPPacketOut(datapath=datapath,
    buffer_id=ofproto.OFP_NO_BUFFER,
in_port=datapath.ofproto.OFPP_CONTROLLER,
    actions=actions, data=p.data)
print "Sending router ID reply: ", ROUTER_ID
 datapath.send_msg(out)
```
Future Work

Our ideas and some questions

- Conduct further large-scale measurements
  - e.g., comparison of SAAMI-generated traceroutes to real traceroute data
  - Congestion estimation
- How to arbitrate access to SAAMI?
- Would providers even allow access to core infrastructure to do this?
We have a paper in progress and would love your feedback!

https://arxiv.org/abs/1702.07946

SAAMI

- New architectural vision for the active measurement infrastructure
- Initial feasibility testing demonstrates promise
- Seeking feedback from the measurement community
Related Work

- Much work involved in measuring OpenFlow processing delays (Rostos, He, others)
- SLAM (Yu et al.), generates custom packets that traverse a path within a datacenter, which themselves trigger control-plane messages to a central controller within a datacenter to compute path latency
- p4 INT (Inband Network Telemetry) – data plane information (e.g. per-hop latency, egress port information, etc) inserted directly into data packets as additional header fields
Calibration

Accounting for Controller-Switch Latency

- Controller measures total time between `OFPktOut` and `OFPktIn` messages
  - Really want time between packet emission by switch and corresponding reply
- Estimate controller to switch latency by calculating time between built-in `OFEchoRequest`-`OFEchoReply` messages for each target
- Subtract estimated controller-switch latency from `OFPktOut`-`OFPktIn` time to obtain RTT estimate
Accounting for Switch Processing Delays

- Switch doesn’t instantaneously emit probe upon receiving a `OFPktOut` – how long does it take?

- Measure time between `OFPktOut` transmission and probe emission from switch

- Measure time between probe receipt and `OFPktIn` message from switch
Calibration

Accounting for Switch Processing Delays

> 0.95 between 1.5 and 2.0 ms time $\Delta$ between `OFPktOut` and packet emission

> 0.95 between .75 and 1.2 ms time $\Delta$ between `OFPktIn` and packet receipt
Accounting for Multiple-Probe \texttt{OFPktOut} Messages

- TCP implementation can cause multiple probes to be “bundled” into one \texttt{OFPktOut} message; must quantify time variation between \texttt{OFPktOut} arrival at switch and bundled probe emission.
- Not a significant source of latency – largest observed delay incurred by a probe less than .5 ms.