

### 

Dataplane testing from *DC to Daylight* Using the Open Traffic Generator API

> Chris Sommers, SW Architect Ankur Sheth, Director of Engineering Keysight Technologies

### **Speaker Biographies**



**Chris Sommers** is a software architect with decades of experience in the design of hardware and software for wireless and wireline networking. Chris works at Keysight Networking Labs where he investigates emerging technologies. Chris also participates in several P4.org working groups. He holds a Bachelor's degree in Electronic Engineering from Cal Poly San Luis Obispo, CA.



**Ankur Sheth** is a member of Product Strategy Group at Keysight where he works on NPIs and also leads Keysight's Network Test Virtualization products. Over the years Ankur has led numerous teams that have built high quality breakthrough products in wired and wireless networking space. Ankur holds a Master's degree in Electrical Engineering (Computer Networks) from University of Southern California, CA.

XIA Chris and Ankur have helped develop the well-known Ixia line of packet testers manufactured by Keysight Technologies.



### I'm a typical P4 Dataplane Developer – What's my workflow ?

Maintenance headache!

- Write some really cool P4 code
- Run it in bmv2 or a vendor P4 simulator (e.g. Tofino Model)
- Throw packets at it with Scapy/PTF, Trex, custom SW tools, etc.
- Test it on *real* P4 switching device (Device Under Test, or DUT)
- Use a hardware-based traffic generator for precision & performance
- Problem can't re-use your scripts! Scapy/PTF doesn't natively support true line-rate testers
   Lots of extra work!
  - Write custom libraries/wrappers for PTF?





### **Traditional Test Environments vs. Speed/Scale**



### **Wish-List for Packet Testing**

As a developer, I want it all, and I want it now:

- Write my test script once, run it everywhere
- Speeds from slow simulators ("DC") up to line rate ("Daylight")
- Model-based, declarative API config as a "document"
- Open standard vendor-agnostic, community ecosystem
- Free, software-based solutions
- Commercial, full-performance solutions
- Easy onramp download and go in minutes





### **Open Traffic Generator – Model-based Packet Testing!**



### Wish-list answered! snappi + OpenTrafficGenerator



### How to demonstrate Snappi?

- As an exercise, I converted an existing PTF test "**demo1**" from Andy Fingerhut's well-known p4-guide repo: <u>https://github.com/jafingerhut/p4-guide</u>
- I forked the repo: https://github.com/chrispsommers/p4-guide/tree/snappi-tests2 and made a new test directory demo1-snappi
- I modified one of the existing tests *FwdTest* and replaced Scapy with snappi
- I added several other tests to illustrate other snappi techniques and features
- Today's demo will showcase the added snappiFwdTest4PortMesh test which does some cool things:
  - Sets up 4 Ixia-c traffic engines + one Ixia-c controller (5 Docker containers)
  - Configures 12 flows to establish a 4x4 port mesh.
    - Each flow includes auto-increment in the IP dest address to cover a /24 subnet
    - · Send 100-byte packets at a precise 50 PPS using the built-in Ixia-c scheduler
  - Retrieve all per-port and per-flow Tx and Rx statistics and compare to expected values
  - Retrieve all captured packets, examine the IP addresses and confirm the complete port-mesh was received.

Note this same test could easily be enhanced to control line-rate testers, at full speed and scale, by changing a few parameters



### Demo Setup: PTF Test using bmv2, snappi+lxia-c Tgen



### snappi code snippet – creating 12 flows

```
i = 0
                                                                     dst1
for src in self.port ndxs:
    for dst in self.port ndxs:
        if src == dst:
            continue # no hairpin switching
        print("Configuring flow[%d]: %s => %s" % (i, ports[src].name, ports[dst].name))
        flow = self.cfg.flows.flow(name='port%d-%d' %(src+1, dst+1))[-1]
                                                                                                 Instantiate a flow
        # flow endpoints
        flow.tx rx.port.tx name = ports[src].name
        flow.tx rx.port.rx name = ports[dst].name
        # configure rate, size, frame count
                                                    Precision scheduler: PPS, BPS or % line rate
        flow.size.fixed = 100
        flow.rate.pps = 50
        flow.duration.fixed packets.packets = self.tx count
        # configure protocol headers with defaults fields
        flow.packet.ethernet().ipv4().tcp()
                                                                  Scapy Equivalent: Ether()/TCP()/IP()
        eth = flow.packet[0]
        eth.src.value = host macs[src]
        eth.dst.value = host macs[dst]
        ipv4 = flow.packet[1]
        ipv4.dst.increment.start = ip hosts[dst]
        ipv4.dst.increment.step = '0.0.0.1'
                                                                 Auto-increment a packet header- built-in to Snappi!
        ipv4.dst.increment.count = self.tx count
        ipv4.src.value = ip hosts[src]
        ipv4.time to live.value = 64
        tcp = flow.packet[2]
        tcp.src port.value = 1234
        tcp.dst port.value = 80
```



i + = 1

### Test Outline - snappiFwdTest4PortMesh

- Configure Ixia-c for 12 traffic flows, into veth2, veth4, veth6 and veth8 (dataplane ports 1-4 respectively in the P4 code). The 12 flows comprise a full-mesh, full-duplex test of port forwarding. Each flow will send 256 packets into its port, incrementing the last byte of the DIP from 1 to 256. Each flow will emit packets at 50 packets per second. We wait until the pipeline finishes processing all packets (or timeout while waiting).
- · Configure Ixia-c to capture all the return traffic
- Start the traffic flows and capture the results
- Verify no packets were captured because the P4 dataplane forwarding tables have not been programmed: the default action is drop.
- Configure the P4 tables to match on the DIPs as configured in the traffic flows and forward to the correct egress ports, also performing MAC rewrite.
- Start traffic flow a second time and capture everything. We wait until the received packet counts match the expected values on all flows (or timeout waiting).
- Verify several expectations:
  - Each port transmits the correct count of packets (3\*255, i.e. 255 packets to each of the other ports in the mesh)
  - Port Tx stats = port Rx stats = 3\*255
  - The captured results has the correct number of packets for each flow (same as transmitted to each flow)
  - Examine each captured packet, extract IP src and dest address, and confirm exactly one packet was sent between each "host" and each of 255 "destinations" on the other port's subnets.

#### Demo available at https://github.com/chrispsommers/p4-guide/tree/snappi-tests2

Thanks to Andy Fingerhut for his p4-guide repo at https://github.com/jafingerhut/p4-guide



### Run the Demo!

1.5022



### **Demo1 – Screen Layout and Commands**

chris@chris-VirtualBox: ~/p4-guide/demo1-snappi – 🗆 😣						
chris@chris-VirtualBox: ~/p4-quide/demo1-snappi 112x56	R chris@chris-VirtualBox: ~/p4-quide/demo1-snappi 89x9					
hris@chris-VirtualBox:~/p4-quide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-quide/demo1-snappiS					
chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$					
<pre>chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$</pre>	chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$					
chrisechris-VirtualBox:~/p4-quide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chrisochris-VirtualBox:~/p4-guide/demo1-snappiS					
chrisechris-VirtualBox:~/p4-guide/demoi-snappis	chris@chris-VirtualBox:~/p4-guide/demo1-snappis					
hrischris-Virtual Box: ~/p4-quide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-quide/demo1-spappis					
hris&chris-VirtualBox:/pd-guide/demo1-snappi\$	chris@chris.VirtualBox:~/p4-guide/demo1-snappiS ./run_switch.sh					
chrisechris-VirtualBox:~/p4-quide/demo1-snappi\$	······································					
hrisochris-Virtual Box: ~/o4-ouide/demo1-snappis						
chrisechris-VirtualBox:~/p4-guide/demoi-snappis	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
hrischris-Virtual Box: ~/p4-quide/demo1-snappis	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
hrisochtis-Victual Box: ~/o4-ouide/demo1-snappis	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
hr isochris-Virtual Box: «/h-quide/demol-snappis	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
his social social social social states and stat	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
his source of the second state of the second s	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
his social set is the lock in the set of denotes and provide set of the set o	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
her course of the star and a star defection is applied	chris@chris-VirtualBox:~/p4-guide/demo1-athenaS					
he could be a set to the box - p - p - guide denot - shape is	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
he is so in the second se	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$ sudo tcpdump -eni veth2					
hi togeni ts-vii tuatbox:~/p4-gutde/demot-shappis						
In rigen ris-virtual BOX:~/p4-guide/demoi-snappis	H≠         chris@chris-VirtualBox: ~/p4-guide/demo1-athena 89x10					
ht sector is ver tudebox: ~/p4-gutde/demot-shappis	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
hr togenrus-vurtualbox:~/p4-guide/demoi-shappis	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
hrisgenrus-vurtualbox:~/p4-guide/demoi-snappis	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
chris@chris-virtualBox:~/p4-guide/demoi-snappis	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
chris@chris-virtualBox:~/p4-guide/demoi-snappis	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
chris@chris-VirtualBox:~/p4-guide/demoi-snappis	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
chris@chris-VirtualBox:~/p4-guide/demol-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	<pre>chris@chris-VirtualBox:~/p4-guide/demo1-athena\$ sudo tcpdump -eni veth4_</pre>					
chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$						
chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	He Chris@chris-VirtualBox: ~/p4-guide/demo1-athena 89x10					
chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
:hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
:hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$ sudo tcpdump -eni veth6					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris.VictualRov: /p4.quida/demo1-those 20v10					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	Consecutions-VirtualBox: -/p4-guide/demoir-atnena 89x10					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	christen is-virtuatexx/~/p4-guide/demoi-athenas					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	christen is-virtuatex:-/p4-guide/demoi-athenas					
hris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chrisgich is virtual Box: ~/p4-guide/demoi-athenas					
hris@chris-VirtualBox:~/p4-quide/demo1-snappi\$	chrisgenris-VirtualBox:~/p4-guide/demoi-athenas					
chris@chris-VirtualBox:~/p4-guide/demo1-snappi\$	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
chris@chris-VirtualBox:-/p4-guide/demo1-snappis	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
bris@chris_VirtualBox:~/p4-guide/demo1-snappis	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
hrisAchris-VirtualBox: //p4-guide/demo1-spappiS	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
	chris@chris-VirtualBox:~/p4-guide/demo1-athena\$					
III LSOCHT LS-VLT LUGLDOX; ~/ D4-OULGE/GENO1-SHADDLS						

### Demo1 – snappiFwdTest4PortMesh Results

					chris@chris-Vir	rtualBox: ~/p4-guide/	demo1-snappi _ 🗆 🛛
		chris@chris-Vir	tualBox: ~/p4-guide	/demo1-snappi 112x56			
flow stat for stats port stat flow stat	ts s to settle ts ts						Adding interface veth2 as port 1 Adding interface veth4 as port 2 Adding interface veth6 as port 3 Adding interface veth8 as port 4
							Adding interface veth10 as port 5 Adding interface veth12 as port 6 Adding interface veth14 as port 7
rt 1	Tx Frames	Tx Bytes	Rx Frames	Rx Bytes	Tx FPS	Rx FPS	Server listening on 0.0.0.0:9559
t1 +2	765	76500	765	76500	0	0	
t2	765	76500	765	76500	0	0	₽ chris@chris-VirtualBox: ~/p4-guide/demo1-athena 89x10
t4	765	76500	765	76500	õ	õ	: 192.168.1.1.1234 > 192.168.4.255.80: Flags [none], seq 0:42, win 0, length 42: HTTP
							09:35:38.770898 dd:00:00:00:00:01 > ee:00:00:00:01 ethertype IPv4 (0x0800), length 96 : 192.168.2.1.1234 > 192.168.1.254.88: Flags [none], seq 0:42, win 0, length 42: HTP 09:35:38.773273 dd:00:00:00:00:01 > ee:00:00:00:01, ethertype IPv4 (0x0800), length 96 : 192.168.2.1.1234 > 192.168.1.255.80: Flags [none], seq 0:42, win 0, length 42: HTP 09:35:38.782108 dd:00:00:00:00:1 > ee:00:00:01, ethertype IPv4 (0x0800), length 96
Flow	Rx I	Frames	Rx Bytes	Max latency (ns)	Avg Latency	/ (ns)	: 192.108.3.1.1234 > 192.108.1.255.80: FLags [none], seq 0:42, win 0, length 42: HITP
port1-2		255	25500	35453840	46	501885	09:55:58.780144 00:00:00:00:00:01 > ee:00:00:00:00:01, etnertype 1PV4 (0X0800), tength 40
port1-3		255	25500	41778940	60	030705	
nort2-1		255	25500	47561180	56	501407	chris@chris.VictualRov: /o4.guido/domo1.athona.80v10
port2-3		255	25500	43944880	65	504447	1 $1$ $1$ $1$ $1$ $1$ $1$ $1$ $1$ $1$
port2-4		255	25500	40257680	70	068737	9:35:38.769809 ee:00:00:00:00:00:20 > ee:00:00:00:00:3. etertype IPv4 (0x0800). length 96
port3-1			25500	40677340	64	166530	: 192.168.2.1.1234 > 192.168.3.255.80: Flags [none], seq 0:42, win 0, length 42: HTTP
port3-2		255	25500	39745560	65	531734	09:35:38.769813 ee:00:00:00:00:02 > ee:00:00:00:00:04, ethertype IPv4 (0x0800), length 96
port3-4		255	25500	43394740	7733077		: 192.168.2.1.1234 > 192.168.4.255.80: Flags [none], seq 0:42, win 0, length 42: HTTP
port4-1		255	25500	36029660	58	358524	09:35:38.783352 dd:00:00:00:00:02 > ee:00:00:00:02, ethertype IPv4 (0x0800), length 96
port4-2		255	25500	51298100	74	134157	: 192.168.3.1.1234 > 192.168.2.255.80: Flags [none], seq 0:42, win 0, length 42: HTTP
							: 192.168.4.1.1234 > 192.168.2.255.80: Flags [none], seq 0:42, win 0, length 42: HTTP
t & flow	statistics						
h port tr	ransmits 255*	3 = 765 packet:	s				: 192.168.2.1.1234 > 192.168.1.255.80: Flags [none], seq 0:42, win 0, length 42: HTTP
& rx port	t stats are io	dentical					09:35:38.769809 ee:00:00:00:00:02 > ee:00:00:00:00:03, ethertype IPv4 (0x0800), length 96
h Rx flow	w received 25	5 packets					: 192.168.2.1.1234 > 192.168.3.255.80: Flags [none], seq 0:42, win 0, length 42: HTTP
tured pag	cket contents						09:35:38.769813 ee:00:00:00:00:00:00:00:00:00:00:00:00:0
ure from	port port1						: 192.108.2.1.1234 > 192.108.4.255.80: Flags [none], seq 0:42, Win 0, Length 42: HITP
ure from	port port2						(-102, 168, 3, 1, 1234, 5, 192, 168, 2, 255, 80, 1235, 50, 50, 50, 50, 50, 50, 50, 50, 50, 5
ure from	port port4						09:35:38.787112 d:00:00:00:00:02 > e:00:00:00:02 = 0 + + + + + + + + + + + + + + + + + +
plete IP	address mesh	was received.					: 192.168.4.1.1234 > 192.168.2.255.80: Flags [none], seq 0:42, win 0, length 42: HTTP
19.768s C traffic engines and controller					1 192.168.2.1.1234 > 192.168.4.255.80: Flags [none], seq 0:42, win 0, length 42: HTTP 09:35:38.784770 div00:04 > esc:06:080.06:081.06:04 > esc:02.04 > esc:02.0		
irtualBox		demo1-snappi\$					



Fetching all Done waiting Fetching all Fetching all Port Stats

Flow Stats

Perifying po - Verify ea - Verify ea verify ea verifying ca vetching cap vetching cap vetching cap vetching cap - Verify co

Ran 1 test i

Killing Ixia 559a37175bc8 21c22bf71ddb 2521d552369a 1c4e58e2fe4a

17906ad2cb12 hris@chris-

### snappi vs. Scapy

ltem	snappi	Scapy
Portability	One script to cover SW and HW traffic generators	Only works for SW generator. Migration to HW = start over
Scalability (# ports)	<ul> <li>Add CPU cores (SW) or line cards/chassis (HW) to scale-out to any size</li> <li>Controller and Traffic Engines/Line cards talk over sockets, easy to scale-out</li> </ul>	<ul> <li>Limited by #NICs; can use fanout switches with additional ad-hoc automation – cumbersome!</li> </ul>
Automation	snappi and/or REST automation, designed for automation	Pytest or PTF; framework and dataplane are synonymous
Performance/ Throughput	<ul> <li>HW Traffic-generators: 400Gbps port speeds</li> <li>Ixia-c SW traffic generator: 10Gbps @ 64 byte frame size using one Xeon class core, approaching 100Gbps depending upon packet length; uses DPDK</li> </ul>	<ul> <li>Limited by CPU speed – "slow-ish"</li> <li>Not optimized for DPDK</li> </ul>
CPU Resources (SW Traffic Gen)	<ul> <li>1 core (shared) for controller</li> <li>Hi-performance: 1 core per direction, per port</li> <li>Best-Effort: 1 core per direction (shared among ports)</li> </ul>	<ul><li>Unclear, but scapy believed to run on one core.</li><li>Roll your own multi-core version for higher performance</li></ul>
Scheduler	<ul> <li>Precision scheduling of multiple flows, nsec precision in HW</li> <li>Specified as packets-per-second, bits-per-second, % of max line rate</li> <li>Burst Mode</li> </ul>	Roll-your-own scheduler – good luck!
Packet types	• Ethernet, VLAN, GRE, GTPv1, GTPv2, IPv4, IPv6, ICMP, ICMPv6, UDP, TCP, custom. More protocols are on the way.	Essentially unlimited, easy to design new ones
Header patterns	• Built-in patterns like increment, decrement, list, etc. directly executed by the engines at speed, to generate millions of unique packets.	<ul> <li>Hand-build your own patterns in python, "manually" send them in sequence.</li> </ul>
Flow tracking	Built-in "instrumentation" header tracks up to 256 unique flows per port	Not supported
Statistics	<ul> <li>Per-port and per-flow transmit and receive statistics</li> <li>Latency measurements (min, max, avg) per flow</li> </ul>	Roll your own!
Capture	<ul> <li>Capture packets with filters, access via port or flow</li> <li>Process in memory, write to Pcap file, send to tcpdump</li> </ul>	Capture packets per port
Statefulness/ Interactivity	Not supported	Good support, easy to perform interactive sessions



### github.com/open-traffic-generator

Open Traffic Generator         Repositories       7       Packages       A People       Projects									
Pinned repositories									
県 models Open Traffic Generator models ● Python ☆ 10 왕 3	<mark>,                                    </mark>	및 Ixia-c Ixia-c Traffic Generator ☆ 1							
□       ixnetwork         The Keysight IxNetwork implementation of the open-traffic-generator models.         ● Python       ☆ 4       % 1	<ul> <li>snappi-tests</li> <li>End-to-end test scripts written in snappi</li> <li>Python</li> </ul>								

Find us on GitHub!

- Use Ixia-c & snappi
- Report bugs
- Log feature requests
- Contribute to models
- Contribute other backends





## Thank You

chris.sommers@keysight.com ankur.sheth@keysight.com <u>https://github.com/open-traffic-generator</u>

# **BACKUP SLIDES**



### **Flow-Tracking, Instrumentation Explained**

Ixia packet testers utilize a proprietary flow-tracking technique which involves inserting a special "instrumentation header" into the packet. It gets inserted after the last valid protocol header, i.e. it forms the first portion of "payload." This header, which is decribed in the ptf/scapy\_contrib/ixia\_scapy.py Scapy file, contains several interesting fields:

- 12-byte fixed "signature" which serves as a marker to indicate start of header
- 4-byte PGID or "port group ID" field; think of this as a flow ID
- 32-bit sequence number which can be used to detect packet drops



This technique was originally pioneered to enable hardware-based testers to perform real-time analysis of line-rate traffic prior to economically-viable protocol parsing engines (like P4 ASICs). The same technique can be done in CPUs (Athena) at lower speeds (approaching 100Gbps for larger packet sizes, limited by the packet-per-second rate).



### Use-case: CI/CD Pipeline End-to-End Test using snappi



### Use-case: SONiC PFC test using + snappi + Ixia tester



