In-band Network Telemetry via Programmable Dataplanes

Changhoon Kim*, Anirudh Sivaraman**, Naga Katta***, Antonin Bas*, Advait Dixit*, Lawrence J Wobker*
*Barefoot Networks, **Massachusetts Institute of Technology, ***Princeton University
{chang, antonin, adixit, ljw}@barefootnetworks.com, anirudh@csail.mit.edu, nkatta@cs.princeton.edu

ABSTRACT

In-band Network Telemetry (INT) is a new abstraction that allows data packets to query switch-internal state such as queue size, link utilization, and queuing latency. We prototype INT in the recently proposed P4 language using a software switch as the implementation platform. We then show how INT can be used to diagnose performance problems such as intermittent latency spikes. Instructions to replicate our demo are available at http://git.io/sosr15-int-demo

CCS Concepts

•**Networks** → **Programmable networks;**

Keywords

Programmable forwarding planes; network debugging

1. IN-BAND NETWORK TELEMETRY

Software-Defined Networking (SDN) has been successful because it lets network owners and operators "program" network behavior. SDN's programmability, however, is confined to the network control plane today. The forwarding plane is still largely dictated by fixed-function switching chips. Our goal is to change that, and to allow programmers to define how packets are to be processed all the way down to the wire. This is made possible by a new generation of high-performance forwarding chips. At the high-end, new programmable protocol-independent switch chips promise multi-Tb/s of packet processing [6, 4, 2]. At the mid- and low-end of the performance spectrum, CPUs [7], GPUs [8], FPGAs [1], and NPUs [3] already offer great flexibility with performance of a few tens to hundreds of Gb/s.

In addition to programmable forwarding chips, we need a high-level language to dictate the forwarding behavior in a target-independent fashion. "P4" [5] (www.p4.org) is such a language. In P4, the programmer declares how packets are to be processed, and a compiler generates a configuration for a protocol-independent switch chip or NIC. For example, the programmer might program the switch to be a top-of-rack switch, a firewall, or a load-balancer, and might add features to run automatic diagnostics and novel congestion-control algorithms.

In this demo, we will give a brief primer on the P4 language by showing and compiling some example P4 programs to derive various network dataplane personalities, demonstrating the power of writing portable and reusable P4 programs. Then, we will demonstrate INT (In-band Network Telemetry), a powerful new network-diagnostics mechanism

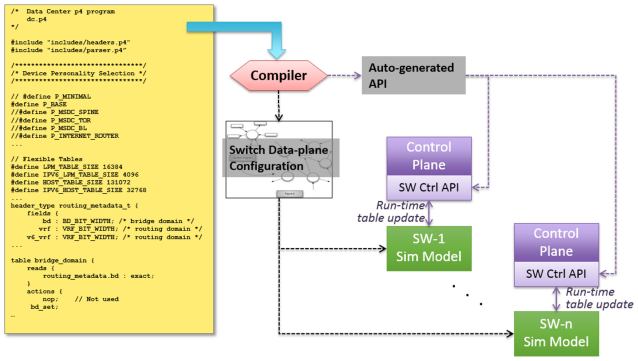

Figure 1: Work flow to compile P4 programs

implemented in P4, inspired by a recent research idea proposing simple and programmable access to switch-internal state [9]. Because INT is implemented in P4, it is runnable on a variety of programmable network devices. With INT, a network owner will be able to debug and root-cause various network incidents rapidly and intuitively. Instructions to replicate our demo are available at http://git.io/sosr15-int-demo

2. DESCRIPTION OF DEMO

We provide an overview of our demo in stages. First, we compile a specific P4 program to each software switch in the topology shown in Figure 2. Then, we setup a periodic HTTP transfer to continuously monitor end-to-end latency. Lastly, we use INT to diagnose the root cause of occasional latency spikes in HTTP transfer latencies.

2.1 P4 development environment

We use the open source P4 switch model available at https://github.com/p4lang/p4factory to create each P4 switch in the topology (Figure 2). Each software switch performs the match-action pipeline processing described by a P4 program. Each software switch also has a control channel that allows the controller to insert, delete, and modify entries in the match-action tables. These APIs are generated automatically by the P4 compiler and provide hooks to perform run-time tasks, such as inserting and removing routes. Our demonstration uses the Mininet [10] environment to route traffic between a number of software P4 switches in a specific topology (Figure 2).

2.2 Continuous HTTP Latency plot

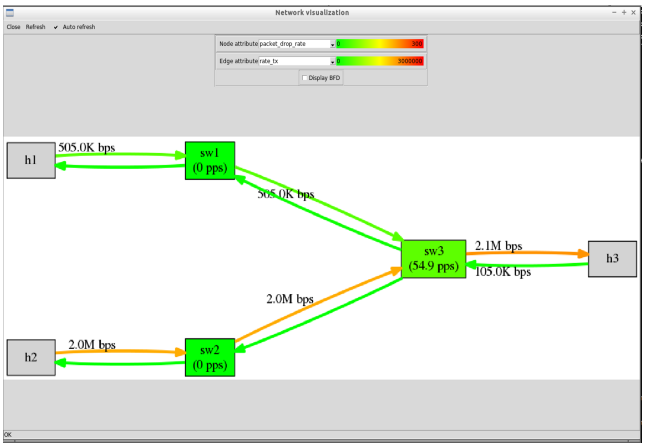

Figure 2: Topology of the experiment with instantaneous throughputs. h1 responds to periodic HTTP requests from h3; h2 occasionally sends a burst of traffic to h3 that interferes with h1's traffic to h3.

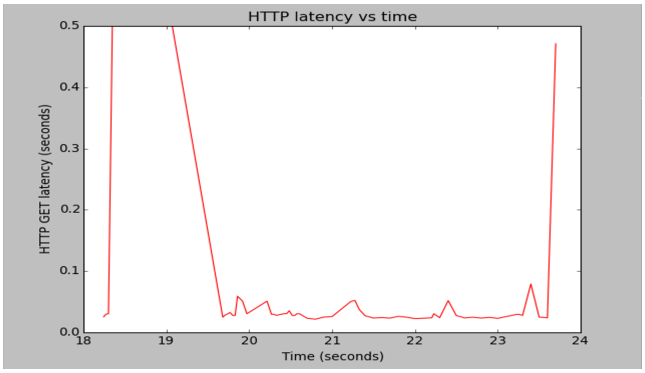

Figure 3: HTTP GET latency time series

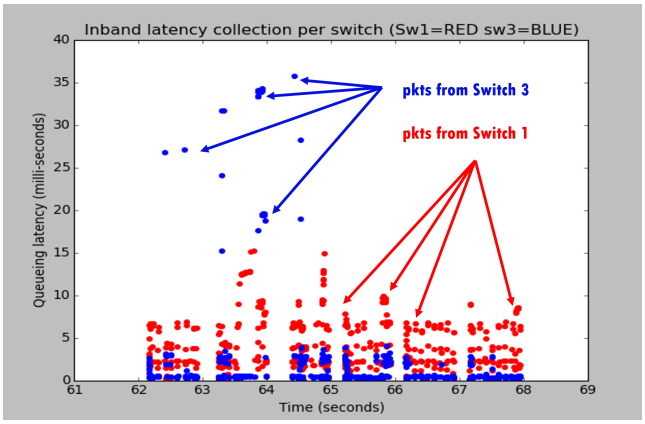

Figure 4: Scatter plot of latency at each switch

We generate a HTTP latency plot (Figure 3) using a constantly looping web request. In the topology, h3 periodically requests a web page from h1 and plots the time it takes to return the page. Periodically, a UDP flow transfers a burst of data from h2 to h3 on the shared path through sw3. This conflict causes latency spikes on the plot.

2.3 Understanding HTTP latency spikes

Each switch in our network uses INT to push telemetry packets periodically that contain the switch ID, as well as the time actually spent in the switch queue into the TCP options field of data packets. When these packets are received at the end host, this data (Figure 4) provides a nearly instantaneous observation of which switch is responsible for the additional latency in the system. Once we have localized the switch with a large queue occupancy (in this case switch 3), we can take further corrective action, such as inspecting what other flows are sharing the same queue on that switch.

3. REFERENCES

[1] The Arista 7124 FX as a High Performance Trade Execution Platform. http://www.argondesign.com/media/uploads/files/P8006-R-001d_The_Arista_FX_Switch_as_an_Execution_Platform.pdf.

[2] Intel FlexPipe. http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-switch-fm6000-series-brief.pdf.

[3] IXP4XX Product Line of Network Processors. http://www.intel.com/content/www/us/en/intelligent-systems/previous-generation/intel-ixp4xx-intel-network-processor-product-line.html.

[4] XPliant™Ethernet Switch Product Family. http://www.cavium.com/XPliant-Ethernet-Switch-Product-Family.html.

[5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[6] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN. In *SIGCOMM*, 2013.

[7] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *SOSP*, 2009.

[8] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: A GPU-accelerated Software Router. In *SIGCOMM*, 2010.

[9] V. Jeyakumar, M. Alizadeh, Y. Geng, C. Kim, and D. Mazières. Millions of Little Minions: Using Packets for Low Latency Network Programming and Visibility. In *SIGCOMM*, 2014.

[10] B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.