Low-latency Network Monitoring via Oversubscribed Port Mirroring

Jeff Rasley[†], Brent Stephens[‡], Colin Dixon[*], Eric Rozner[*],
Wes Felter[*], Kanak Agarwal[*], John Carter[*], Rodrigo Fonseca[†]
[†]Brown University [‡]Rice University [*]IBM Research–Austin,TX

Introduction Modern networks operate at a speed and scale that make it impossible for human operators to manually respond to transient problems, e.g., congestion induced by workload dynamics. Even reacting to issues in seconds can cause significant disruption, so network operators overprovision their networks to minimize the likelihood of problems.

Software-defined networking (SDN) introduces the possibility of building autonomous, self-tuning networks that constantly monitor network conditions and react rapidly to problems. Previous work has demonstrated that new routes can be installed by an SDN controller in tens of milliseconds [10], but state-of-the-art network measurement systems take hundreds of milliseconds or more to collect a view of current network conditions [1, 2, 3, 4, 11]. To support future autonomous SDNs, a much lower latency network monitoring mechanism is necessary, especially as we move from 1 Gb to 10 Gb and 40 Gb links, which require 10x and 40x faster measurement to detect flows of the same size. We believe that networks need to, and can, adapt to network dynamics at timescales closer to milliseconds or less.

This paper introduces Planck, a network measurement architecture that provides statistics on 1 Gb and 10 Gb networks at 3.5–6.5 ms timescales, more than an order of magnitude improvement over the state-of-the-art. Planck does so with a novel use of the well-known port mirroring mechanism: it mirrors all traffic traversing a switch to a small number—one in our current system—of oversubscribed monitor ports. When more traffic is sent to the monitor ports than they can handle, traffic is dropped, resulting in the monitor port emitting a "random" sample of all traffic on the switch. As this is done at line rate in the data path, Planck collects orders of magnitude more samples per second than is possible using previous monitoring mechanisms. In addition to supporting sFlow-style [9] sampling, Planck provides extremely low latency link utilization and flow rate estimates as well as alerts when congestion is detected.

A Planck *collector*, consisting of software running on a commodity server, receives these samples and performs lightweight analysis to provide measurement data to those interested, e.g., an SDN controller.

Design Planck consists of three main components which can be seen in Figure 1: (i) switches configured to provide samples at high rates, (ii) a set of collectors which process those samples and turn them into events and queryable data, and (iii) a controller which can act on those events and data. This paper focuses on the first two.

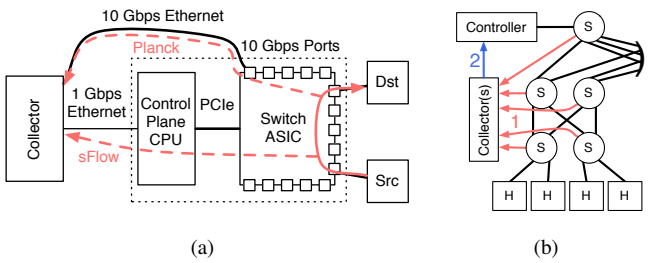
Fast Sampling at Switches Planck is inspired by sFlow [9], which is designed to randomly sample and forward packets and provide them to a collector in real time. However, sFlow samples are typically processed by switch control planes which limit the rate of samples as show in Figure 1(a). For example, the IBM G8264 produces at most 350 samples per second, and thus obtaining an accurate view of the network takes seconds or longer. Counter polling, as in OpenFlow, also uses the control plane and suffers similar problems.

We overcome this by leveraging the port mirroring feature found in most commodity switches today that is traditionally used for diagnosing problems. Port mirroring allows for all—or a subset—of traffic received on a given input port to be both forwarded normally and also copied and sent out a *mirror port*. Importantly, this is done at line-rate in the data plane allowing for as many mirrored packets as the mirror port has bandwidth to carry.

We repurpose this functionality to efficiently produce samples by oversubscribing the mirror port(s). Oversubscription causes the output buffer for the mirror port to eventually fill and then drop packets, constraining the samples to the bandwidth of the output port. We typically allocate 1 port of an N port switch to be the mirror port and the remaining $N - 1$ ports to handle data traffic. This gives a sampling rate expected to be about 1 in $N - 1$.

To study the latency of the samples received by the collector we ran experiments under low and high congestion. The low congestion experiment sent a single line-rate TCP flow and measured the latency between when tcpdump reported the packet sent and when the collector received it. This latency was between 75–150 µs. The high congestion experiment introduced 2 additional line-rate flows that do not contend for resources except the mirror port. This caused the mirror port to become congested and fill its buffer space, which maximized the latency of our samples. Figure 2 shows a delay of about 3.5 ms on an IBM Rackswitch G8264 (10 Gb) and about 6 ms on a Pronto 3290 (1 Gb) switch.

Collector The collector has four goals: (i) process sampled packets at line rate, (ii) infer the input and output ports for each packet, (iii) determine flow rates and link utilization, and (iv) answer queries about the state of the network.



(a) (b)

Figure 1: The architecture of Planck: (a) a switch showing paths for samples using Planck vs sFlow (in red). (b) The network architecture showing one pod of a fat tree with (1) switches providing samples to collectors and (2) collectors processing samples and sending events to the controller.

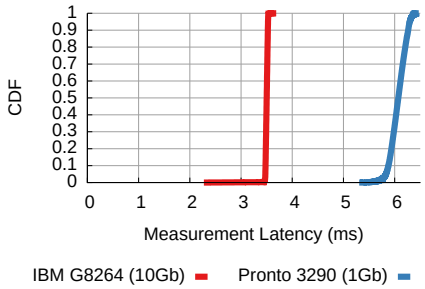


Figure 2: An experiment showing the latency between when a packet is sent and received by the collector on 10Gb and 1Gb switches during high congestion.

The collector uses netmap [8] for line-rate processing and borrows from a substantial body of work on line-rate packet processing on commodity servers [5, 6, 7], so only the last three goals are discussed in detail here.

Determining Input and Output Ports Traditional network sampling techniques produce packet samples with additional metadata, including the input and output ports of the packet. These are particularly important for determining if a given port, and thus link, is congested. Because no additional metadata is added to mirrored packets, the collector must infer the input and output ports from the packet alone. Planck solves this problem by having the controller share the topology of the network and the switches' forwarding table state with the collectors. Our Planck-enabled controller uses deterministic routing, which allows the full path—and thus input and output ports—of a given packet to be inferred from its source and destination. Note that ECMP is deterministic if the hash function is known.

Determining Flow Rates and Link Utilization Traditionally, sampling-based measurement determines flow rates and link utilization by multiplying the throughput of samples received for a given flow or port by the sampling rate. The collector already infers the input and output port for each packet, associates packets with flows, and tracks information about each flow. However, the collector must also infer the sampling rate or otherwise compute flow rates.

The collector does this by using TCP sequence numbers to directly estimate TCP flow throughput. This approach works for all traffic with sequence numbers, not just TCP. If sequence numbers are for packets instead of bytes, we multiply by average packet size. We hope to extend this to traffic without sequence numbers by using the traffic with sequence numbers to estimate the sampling rate and then using the sampling rate to compute estimated bandwidth of the remaining flows. Lastly, to compute link utilization, we simply sum the throughput of all the flows traversing a given link.

Answering Queries The collector currently supports three queries. First, it provides the flows on each link, their current rates, and link utilization described above. Second, it provides full traces for the recent samples it has received, limited by available memory. Third, it allows for subscription to events, e.g., when link utilization crosses a threshold.

Discussion Buffering at the mirror port increases sample latency and steals buffer space that other ports might use to handle bursts. Although our experiments show that this doesn't significantly impact loss or throughput, we hope to be able to reduce or eliminate this buffering in the future. Additionally, determining the sampling rate for traffic without sequence numbers depends on the switch's drop policy, which we are investigating. Our current collector can consume 10 Gb of samples with a single core and we are exploring its scalability further. Finally, we used Planck to implement Hedera-like [1] traffic engineering that detects and reroutes flows in under 10 ms.

References

[1] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI*, 2010.
[2] T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *CoNEXT*, 2011.
[3] A. R. Curtis et al. DevoFlow: Scaling Flow Management for High-Performance Networks. In *SIGCOMM*, 2011.
[4] N. Farrington et al. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. In *SIGCOMM*, 2010.
[5] Intel® DPDK: Data Plane Development Kit. http://www.dpdk.org.
[6] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ToCS*, 18(3):263–297, August 2000.
[7] PF_RING: High-Speed Packet Capture, Filtering and Analysis. http://www.ntop.org/products/pf_ring/.
[8] L. Rizzo. Netmap: A Novel Framework for Fast Packet I/O. In *USENIX ATC*, 2012.
[9] sFlow. http://sflow.org/about/index.php.
[10] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter. PAST: Scalable Ethernet for Data Centers. In *CoNEXT*, 2012.
[11] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter. OpenSample: A Low-latency, Sampling-based Measurement Platform for SDN. Technical Report RC25444: https://ibm.biz/BdRfxB, IBM, 2014.