Demonstration of the Marple System for Network Performance Monitoring

Vikram Nathan[1], Srinivas Narayana[1], Anirudh Sivaraman[1], Prateesh Goyal[1],
Venkat Arun[2], Mohammad Alizadeh[1], Vimalkumar Jeyakumar[3], Changhoon Kim[4]
[1] *MIT CSAIL* [2] *IIT Guwahati* [3] *Cisco Tetration Analytics* [4] *Barefoot Networks*

ABSTRACT

We demonstrate *Marple [15]*, a system that allows network operators to measure a wide variety of performance metrics in real time. It consists of a performance query language, Marple, modeled on familiar functional operators like map, filter, and groupby. Marple is supported by a programmable key-value store on switches, which can compute flexible aggregated statistics (*e.g.,* per-flow counts, moving averages over queueing latencies) over packets at line rate. Our switch design implements performance queries which could previously run only on end hosts, while utilizing only a modest fraction of switch hardware resources. To demonstrate the utility of Marple, we compile Marple queries to a P4-programmable software switch running within Mininet. We demonstrate two example use cases of Marple: diagnosing the root cause of latency spikes and measuring the flowlet size distribution.

CCS CONCEPTS

• **Networks** → **Network monitoring**; *Programmable networks*;

KEYWORDS

Network measurement; network hardware; network programming

ACM Reference Format:
Vikram Nathan, Srinivas Narayana, Anirudh Sivaraman, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Demonstration of the Marple System for Network Performance Monitoring. In *Proceedings of SIGCOMM Posters and Demos '17, Los Angeles, CA, USA, August 22–24, 2017,* 3 pages.
https://doi.org/10.1145/3123878.3131985

1 INTRODUCTION

Effective *performance monitoring* of large networks is crucial to quickly localize problems, such as high queueing latencies, TCP incast, and load imbalance. A common approach to network monitoring is to collect information from endpoints [9, 14, 17, 19] or end-to-end probes [10] to diagnose performance problems. While endpoints provide application context, they lack visibility to localize performance problems in links deep in the network.

Switch-based monitoring could allow operators to diagnose problems with direct visibility into performance statistics. However, traditional switch mechanisms like sampling [1, 5] and mirroring [11, 20] are inadequate. They miss events of interest, since it is infeasible to collect information on all packets. Additionally, neither mechanism provides relevant performance data, like queueing delays.

Upcoming technologies recognize the need for better performance monitoring using switches. In-band network telemetry [3] writes queueing delays experienced by a packet on the packet itself, allowing endpoints to localize delay spikes. The flow cache in the Tetration switching chip [2] provides some flow-level performance metrics. The exposed metrics are useful, but they are fixed and measured at a fixed granularity. For example, the metrics include flow-level latency and packet size variation, but not latency variation, *i.e.*, jitter.

Operator requirements are ever-changing and redesigning hardware is expensive. Instead of adding fixed-function monitoring piecemeal to switches, we seek performance monitoring *primitives* that can be reused for a variety of operator needs. To this end, we have developed *Marple* [16], a performance query language that is implemented on top of a novel programmable switch hardware primitive. We provide an overview of network performance monitoring using Marple in §2. We demonstrate Marple using 2 use cases in §3: debugging a performance problem, *microbursts* [12], and measuring the flowlet size distribution at a particular switch.

2 MARPLE OVERVIEW

Marple is a high-level query language built on familiar functional primitives like `map`, `filter`, and `groupby`. It provides the abstraction of a stream that contains performance information for *every packet* at *every queue* in the network. Using Marple, programmers can focus their attention on traffic experiencing performance issues (*e.g.,* get packets with high queueing latencies), flexibly aggregate information across packets (*e.g.,* compute a moving average over queueing latency per flow), and detect simultaneous performance conditions in the network (*e.g.,* detect when the queue depth is large and the number of connections using the queue is high). Marple also allows composing multiple queries to compute more complex statistics.

Figure 1 gives an overview of Marple. The Marple compiler translates queries to switch programs, which run on programmable switches augmented with a new *programmable key-value store* primitive in hardware [16] to implement per-flow statistics such as moving averages. The Marple paper [16] provides a detailed description of Marple's language, compiler, and hardware. Here, we focus on demonstrating Marple's utility through Mininet [13] case studies.

3 DEMONSTRATIONS

We demonstrate Marple with two performance monitoring examples. The examples run on an emulated Mininet [13] network with the P4 behavioral model [4] software switch. The Marple compiler emits P4 code [8] from the Marple queries to configure the switches.

SIGCOMM Posters and Demos '17, August 22–24, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5057-0/17/08.
https://doi.org/10.1145/3123878.3131985

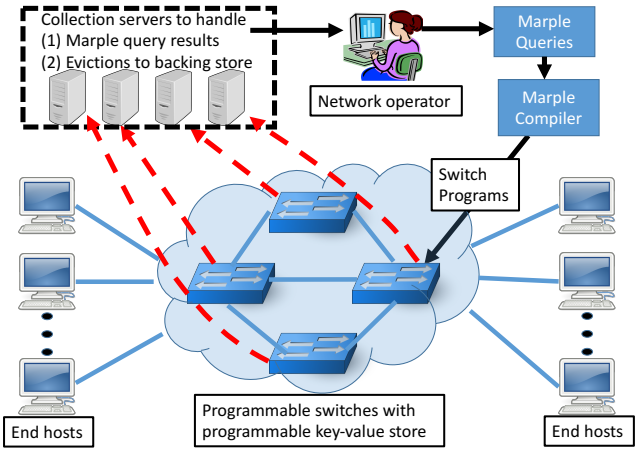


Figure 1: Overview of Marple components.

3.1 Debugging Latency Spikes

We show how to use Marple to diagnose the root cause of latency spikes in recurring HTTP GET requests. The network topology consists of four hosts and two switches in a dumbbell topology (Figure 2). Host `h2` repeatedly downloads a 1 MB object over HTTP from `h1`. Meanwhile, `h3` sends `h4` sporadic bursts of UDP traffic, which `h4` acks. Suppose a network admin notices the irregular latency spikes for the HTTP traffic (Fig. 3a). She suspects a queue buildup and measures the queue depths by writing the query:

```
result = filter(pktstream, srcip == h1 and dstip == h2)
```

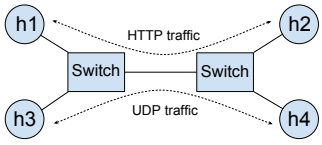
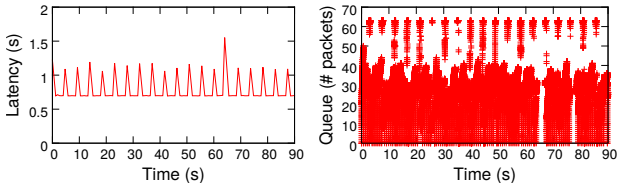


Figure 2: Mininet dumbbell topology used in the demo.

The queue depth for each matching packet is streamed out to a collection server. After plotting the queue latencies, she notices spikes in queue size at egress port 3 on the switch (Fig. 3b) matching the periodicity of the latency spikes. To isolate the responsible flow(s), she divides the traffic into "bursts," which she defines as a series of packets separated by a gap of at least 800ms, as determined from the gap between latency spikes. To do so, she issues the following Marple query:

```
def burst_stats(last_time, nburst, time, pkts, tin):
    if tin - last_time > 800000:
        nbursts++;
        emit();
    else:
        time = time + tin - last_time;
    pkts = pkts + 1;
    last_time = tin;
result = groupby(R1, 5tuple, burst_stats)
```

She runs the query for 72 seconds and sees the result in Figure 4. She concludes, correctly, that UDP traffic between `h3` and `h4` is



(a) HTTP request latency **(b) Queue depth at egress port 2**

Figure 3: Mininet case study measurements.

src → dst	protocol	# Bursts	Time (μs)	# Packets
h3:34573 → h4:4888	UDP	19	8969085	6090
h4:4888 → h3:34573	UDP	18	10558176	5820
h1:1777 → h2:58439	TCP	1	72196926	61584
h2:58439 → h1:1777	TCP	1	72248749	33074

Figure 4: Per-flow burst statistics from Marple.

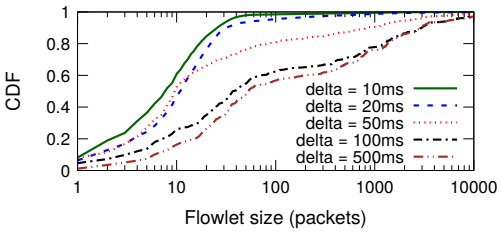


Figure 5: CDF of flowlet sizes for different flowlet thresholds.

responsible for the latency spikes. There are 18 UDP bursts, with an average size of 320 packets and average duration of 472 ms, which matches our emulation setup. Note that the data in Figure 3 and Figure 4 are updated in real time as traffic flows through the switch.

Marple's power and flexibility make this diagnosis simple. End-to-end probes like ping are blind to queue contention on the switch, while switch packet counts alone would have disguised the bursty nature of the offending UDP traffic. Marple's flexible aggregations expose flow-level statistics customized to the problem.

3.2 Flowlet size distributions

We demonstrate how to compute the distribution of flowlet sizes as a function of the flowlet *threshold*, *i.e.*, the time gap above which subsequent packets are considered to be in different flowlets. This analysis has many practical uses, *e.g.,* for configuring thresholds for flowlet-based load balancing strategies [6, 18]. In particular, the performance of LetFlow [18] depends heavily on the distribution of flowlet sizes.

The topology consists of a single switch connecting five hosts: one client and four servers. Flow sizes are drawn from an empirical distribution computed from a trace of a real data center [7]. The switch runs the "flowlet size histogram" query from Figure 7 in [16] for six values of `delta`, the flowlet threshold.

Fig. 5 shows the CDF of flowlet sizes for various values of `delta`. Note that the actual values of `delta` are a consequence of the bandwidth allowed by the Mininet setup; a data center deployment would likely use much lower `delta` values.

REFERENCES

[1] Cisco IOS NetFlow. http://www.cisco.com/c/en/us/products/ios-nx-os-software/

ios-netflow/index.html.
[2] Data center flow telemetry. http://www.cisco.com/c/en/us/products/collateral/data-center-analytics/tetration-analytics/white-paper-c11-737366.html.
[3] In-band Network Telemetry. https://github.com/p4lang/p4factory/tree/master/apps/int.
[4] P4 Behavioral Model. https://github.com/p4lang/behavioral-model.
[5] sFlow. https://en.wikipedia.org/wiki/SFlow.
[6] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese. CONGA: Distributed Congestion-Aware Load Balancing for Datacenters. In *SIGCOMM*, 2014.
[7] Alizadeh, Mohammad. Empirical Traffic Generator. https://github.com/datacenter/empirical-traffic-gen, 2017.
[8] 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 CCR*, July 2014.
[9] M. Ghasemi, T. Benson, and J. Rexford. Dapper: Data Plane Performance Diagnosis of TCP. In *SOSR*, 2017.
[10] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. In *SIGCOMM*, 2015.
[11] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In *NSDI*, 2014.
[12] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker. In-band Network Telemetry via Programmable Dataplanes. In *SOSR demos*, 2015.
[13] 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.
[14] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. Trumpet: Timely and Precise Triggers in Data Centers. In *SIGCOMM*, 2016.
[15] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim. Language-Directed Hardware Design for Network Performance Monitoring. In *SIGCOMM*, 2017.
[16] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim. Language-directed hardware design for network performance monitoring. In *SIGCOMM*, 2017.
[17] P. Tammana, R. Agarwal, and M. Lee. Simplifying datacenter network debugging with pathdump. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 233–248, GA, 2016. USENIX Association.
[18] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall. Let it Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *NSDI*, 2017.
[19] M. Yu, A. Greenberg, D. Maltz, J. Rexford, L. Yuan, S. Kandula, and C. Kim. Profiling network performance for multi-tier data center applications. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 57–70, Berkeley, CA, USA, 2011. USENIX Association.
[20] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, and H. Zheng. Packet-Level Telemetry in Large Datacenter Networks. In *SIGCOMM*, 2015.