Latency in Software Defined Networks: Measurements and Mitigation Techniques

Keqiang He†, Junaid Khalid†, Sourav Das†, Aaron Gember-Jacobson†, Chaithan Prakash†, Aditya Akella†, Li Erran Li*, Marina Thottan*
†University of Wisconsin-Madison, *Bell Labs

ABSTRACT

We conduct a comprehensive measurement study of switch control plane latencies using four types of production SDN switches. Our measurements show that control actions, such as rule installation, have surprisingly high latency, due to both software implementation inefficiencies and fundamental traits of switch hardware. We also propose three measurement-driven latency mitigation techniques—optimizing route selection, spreading rules across switches, and reordering rule installations—to effectively tame the flow setup latencies in SDN.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Network**]: General

Keywords

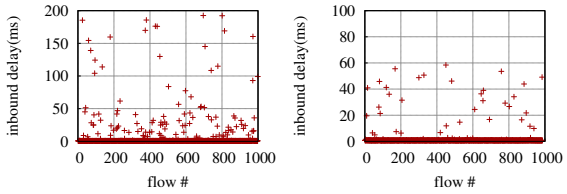
Software-defined Networks; Latency; Measurement; Mitigation

1. INTRODUCTION

Software defined networking (SDN) advocates for the separation of control and data planes in network devices, and provides a logically centralized platform to program data plane state [5]. This has opened the door to rich network control applications that can adapt to changes in network topology or traffic patterns more flexibly and more quickly than legacy control planes [1, 2]. For such applications, timely interaction between the logically central SDN control plane and network switches is crucial. However, it is unknown whether SDN can provide sufficiently responsive control to support the aforementioned applications.

To this end, we present a thorough systematic exploration of latencies in four types of production SDN switches from three different vendors—Broadcom, Intel, and IBM—using a variety of workloads. Key highlights from our measurements are as follows: (1) We find that *inbound latency*, i.e., the latency involved in the switch generating packet events to the controller can be high (8 ms per packet on average on Intel). We find the delay is particularly high whenever the switch is simultaneously processing forwarding rules received from the controller. (2) We find that *outbound latency*, i.e., the latency involved in the switch installing/modifying/deleting forwarding rules, is high as well (3ms and 30ms per rule for insertion and modification, respectively, in Broadcom).

SIGMETRICS'15, June 15–19, 2015, Portland, OR, USA.
ACM 978-1-4503-3486-0/15/06.
http://dx.doi.org/10.1145/2745844.2745880.



(a) with flow_mod/pkt_out (b) w/o flow_mod/pkt_out

Figure 1: Inbound delay on Intel switch, flow arrival rate = 200/s

Some of our findings show that poor switch software design contributes significantly to observed latencies (affirming [4, 6]). We believe that near term work will address these issues; our measurements with an early release of Broadcom's OpenFlow 1.3 software exemplify this. More crucially, our measurements also reveal latencies that appear to be fundamentally rooted in hardware design: e.g., rules must be organized in switch hardware tables in priority order, and simultaneous switch control actions must contend for limited bus bandwidth between a switch's CPU and ASIC. Unless the hardware significantly changes—and our first-of-a-kind in-depth measurement study may engender such changes—we believe these latencies will manifest even in next generation switches.

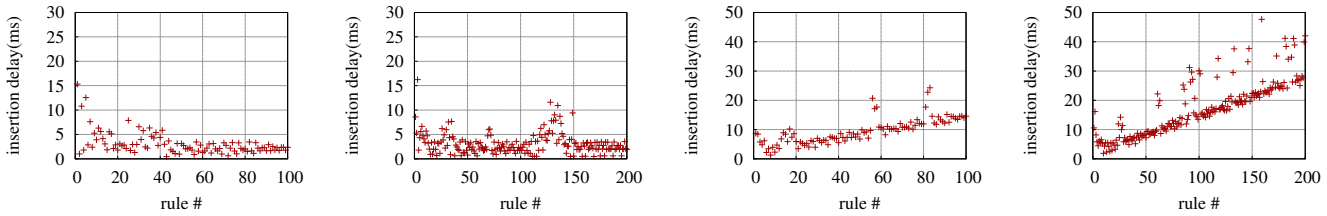
To mitigate the impact of outbound latency, and support the needs of SDN apps, we propose three immediately deployable techniques: flow engineering (FE), rule offload (RO), and rule reordering (RR). Simulation for fast fail-over and responsive traffic engineering applications shows our techniques can improve the time taken to update network state in these scenarios by factors of 1.6-5X. Detailed results can be found in [3].

2. LATENCY MEASUREMENTS

We conduct measurements on four types of switches from three vendors—Broadcom 956846K with OpenFlow 1.0 firmware (BCM-1.0) and OpenFlow 1.3 firmware (BCM-1.3), Intel FM6000 (Intel) and IBM G8264 (IBM). We craft our experiments to ensure the latency impact of various factors can be measured directly from the data plane with the exception of packet_in generation latency.

2.1 Inbound Latency

Representative results for inbound latency are shown in Figure 1a and 1b, respectively, for the Intel FM6000 switch. For the first experiment, we see that the inbound latency is quite variable with a mean of 8.33ms, a median of 0.73ms, and a standard deviation of



(a) burst size 100, same priority (b) burst size 200, same priority (c) burst size 100, incr. priority (d) burst size 200, incr. priority

Figure 2: BCM-1.0 per-rule insertion latency

31.34ms. For the second experiment, the inbound delay is lower (mean of 1.72ms, median of 0.67ms) and less variable (standard deviation of 6.09ms). We also observe that inbound latency depends on the *packet_in* rate: e.g., in first experiment the mean is 3.32 ms for 100 flows/s (not shown) vs. 8.33ms for 200 flows/s (Figure 1a). Our conversations with the switch vendor suggest that the limited bus bandwidth between the ASIC and switch CPU is the primary factor contributing to inbound latency.

2.2 Outbound Latency

We study the outbound latencies for three different *flow_mod* operations: insertion, modification, and deletion. For each operation, we examine the latency impact of key factors, including flow table occupancy and rule priority.

2.2.1 Insertion Latency

Figure 2 shows representative results on BCM-1.0 switch. We observe that: (1) rule complexity does not affect insertion delay; (2) same priority insertions in BCM-1.0, BCM-1.3, Intel and IBM are fast and not affected by flow table occupancy; and (3) priority insertion patterns can affect insertion delay very differently. For Intel, increasing priority insertion is similar to same priority insertion, but decreasing priority incurs much higher delay. For BCM-1.3 and IBM the behavior is inverted: decreasing priority insertion is similar to same priority insertion and increasing priority insertion incurs higher delay. For BCM-1.0, insertions with different priority patterns are all much higher than insertions with same priority. Key root causes for observed latencies are: (1) how rules are organized in the TCAM, and (2) the number of TCAM slices. *Both of these are intrinsically tied to switch hardware.* Even in the best case (Intel), per-rule insertion latency of 1ms is higher than what native TCAM hardware can support. Thus, in addition to the above two causes, there appears to be an *intrinsic switch software overhead* contributing to all latencies.

2.2.2 Modification Latency

We observe that the per-rule modification latency on BCM-1.0 and IBM is impacted purely by table occupancy, not by rule priority structure. Conversations with Broadcom indicated that TCAM modification should ideally be fast and independent of table size, so the underlying cause appears to be less optimized switch software in BCM-1.0. For BCM-1.3 and Intel, the per-rule modification delay is independent of rule priority, table occupancy, and burst size; BCM-1.3's per-rule modification delay is 2X higher than insertion.

2.2.3 Deletion Latency

For BCM-1.0, BCM-1.3, Intel and IBM, deletion latency is not affected by the priorities of rules in the table or the order of deletion. However, deletion delay is affected by table occupancy—it decreases with rule number in all the switches we measured. Thus we infer that deletion is incurring TCAM reordering.

3. MITIGATION TECHNIQUES

We propose three techniques to mitigate the outbound latencies imposed by current switches: *Flow engineering* (FE) leverages our empirical latency models to compute paths such that the latency of installing forwarding state at any switch is minimized. *Rule offloading* (RO) computes strategies for opportunistically offloading installation of some forwarding state to downstream switches. Finally, *rule reordering* (RR) sends rule installation requests in an order that is optimal for the switch in question. By reducing installation latency per switch (FE + RR) and enabling network-wide parallel updates (RO), rule updates can finish much faster. We evaluated our mitigation techniques for three applications: failover in a tunneled WAN, two-level responsive traffic engineering, and MicroTE [2]. Our simulations show we can improve the time taken to update network state in these scenarios by factors of 1.6-5X.

4. CONCLUSION

Our measurements across four OpenFlow-based switches show that the latencies underlying the generation of control messages (pkt_in's) and execution of control operations (flow_mod's) can be quite high, and variable. We find that the underlying causes are linked both to software inefficiencies, as well as pathological interactions between switch hardware properties (shared resources and how forwarding rules are organized) and the control operation workload (the order of operations issues, and concurrent switch activities). Finally, to mitigate the challenges these latencies create for SDN in supporting critical management applications, we present three measurement-driven techniques. Our evaluation shows that these mechanisms can tame flow setup latencies effectively.

Acknowledgement

This work is supported in part by NSF grants CNS-1302041, CNS-1314363 and CNS-1040757.

5. 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] K. He, J. Khalid, S. Das, A. Akella, E. L. Li, and M. Thottan. Mazu: Taming latency in software defined networks. *University of Wisconsin-Madison Technical Report*, 2014.

[4] D. Y. Huang, K. Yocum, and A. C. Snoeren. High-fidelity switch models for software-defined network emulation. In *HotSDN*, 2013.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM CCR*, 2008.

[6] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. Oflops: An open framework for openflow switch evaluation. In *PAM*, 2012.