Prophet: Real-time Queue Length Inference in Programmable Switches

Shuhe Wang[†], Jun Bi[#], Chen Sun[†], Yu Zhou[†]

[†#]Institute for Network Sciences and Cyberspace, Tsinghua University

[†#]Beijing National Research Center for Information Science and Technology (BNRist)

[†]{wangshuh18,c-sun14,y-zhou16}@mails.tsinghua.edu.cn, [#]junbi@tsinghua.edu.cn

CCS CONCEPTS

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

KEYWORDS

Programmable Switches; PISA; Queue Modeling; Flow Scheduling

ACM Reference format:

Shuhe Wang, Jun Bi, Chen Sun, *et al.*. 2019. Prophet: Real-time Queue Length Inference in Programmable Switches. In *Proceedings of SOSR '19: Symposium on SDN Research, San Jose, CA, USA, April 3–4, 2019 (SOSR '19),* 2 pages.
https://doi.org/10.1145/3314148.3318050

1 INTRODUCTION

Programmable switches enable the implementation of many complex network functions directly in the data plane. Protocol Independent Switch Architecture (PISA) is a state-of-the-art architecture for programmable switches [1]. After entering a PISA switch, packets first go through an *ingress pipeline*, then enter the *traffic manager* that maintains multiple queues, and are finally processed by an *egress pipeline*.

However, there exists an intrinsic constraint in PISA. The traffic manager generates metadatas of queue lengths which are only accessible in egress, while the ingress has no visibility in the queue status. This prevents PISA switches from supporting many advanced network functions. For instance, DRILL [3] employs per-packet load balancing by deciding which queue a packet should enter based on the lengths of candidate queues. The decision has to happen in ingress before packet queuing, which cannot be supported in PISA.

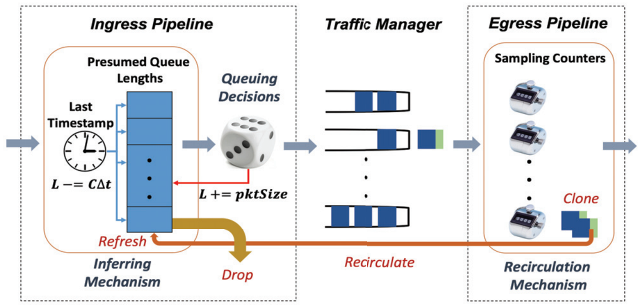


Figure 1: Prophet workflow

To address the above constraint, a straightforward approach is to enable ingress to send periodic *probing packets* to *all queues of all ports* and deliver the queue information from egress through the *recirculation* capability of PISA switches. However, for real-time applications such as DRILL, probing must finish within very short (a few microseconds) intervals. This introduces massive additional packets to be queued, which could cause inflated latency and potential packet drops when queues are already heavily occupied. A potential enhancement is to directly clone original traffic in egress and recirculate them to ingress with the queue information. Doing so could avoid generating additional packets that compete for the queues. Nevertheless, it takes time of microseconds for packets to recirculate, while packet processing rates are usually at a nanosecond timescale, resulting in ingress receiving untimely and inaccurate queue lengths.

In this poster, we present Prophet to enable *real-time and accurate queue length inference in ingress* in PISA-based programmable switches. As ingress knows the target queue of every single packet, instead of waiting for egress to back-propagate queue information to ingress, Prophet actively estimates queue lengths in ingress. Prophet first models FIFO queue behaviour by recomputing the length of a queue whenever a new packet enters based on port bandwidth. Second, Prophet addresses the implementation challenge of supporting multiplication function through approximation. Finally, to eliminate the inference inaccuracy of approximation, Prophet enables egress to periodically back-propagate accurate queue length to ingress. Our evaluation shows that Prophet can maintain accurate queue lengths in egress and could effectively support functions such as DRILL.

This poster is supported by the National Key R&D Program of China (No.2017YFB0801701), the National Science Foundation of China (No.61872426). Jun Bi is the corresponding author.

2 PROPHET DESIGN

We illustrate Prophet design in Figure 1. The key idea of Prophet is inferring queue lengths in ingress. So we maintain a register array in ingress to hold inferred queue lengths. Next we introduce Prophet in detail.

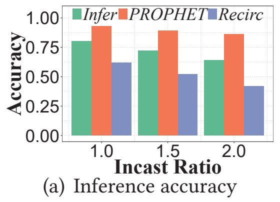
Queue behavior modeling. Prophet models queue behaviour in ingress. By default, all queues in switches are FIFO. Therefore, when a new packet is inserted into a queue, its length increases. On the other hand, it decreases when dequeuing packets. Despite ingress does not know the exact time of packet dequeuing, we could estimate this according to interface throughput. For simplicity, we assume that an interface fetches packets from all of its queues in a round-robin fashion. Therefore, we could estimate the queue length decreasing ratio. For applications like DRILL, queue length information is only needed on packet arrival. Therefore, we trigger queue length calcuation when new packets arrives in ingress. We use register to record the timestamps when packets arrive. The duration between two adjacent packets is calculated as $\Delta t = t_{current} - t_{last}$. The queue length decreasing rate is denoted as C. Therefore, traffic of $\Delta L = C\Delta t$ is dequeued. Note that if ΔL is above the current queue length, then the queue should be empty, and the length is directly set to zero.

Approximating multiplication in PISA switches. Multiplication is not supported in the switch programming language such as P4. To address this challenge, we propose to approximate multiplication with shifts and additions. We observe that shift operations are supported and are equal to multiplication with powers of 2. Therefore, we approximate C with $2^{k_1} + 2^{k_2} + \ldots + 2^{k_t}, k_i \in \mathbb{Z}, i = 1, 2, \ldots, t$. Thus, $\Delta L \approx 2^{k_1}\Delta t + 2^{k_2}\Delta t + \ldots + 2^{k_t}\Delta t$. If $k_i > 0$, then $2^{k_i}\Delta t = \Delta t << k_i$, otherwise $2^{k_i}\Delta t = \Delta t >> (-k_i)$. This way, the multiplication can be substituted with $(t-1)$ shift operations and $(t-1)$ additions with reasonable accuracy.

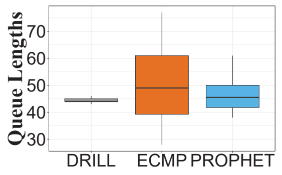
Back-propagating queue lengths for high accuracy. Although the approximation is accurate in a short time, the bias still exists and can be accumulated in the long run, which significantly compromises inference accuracy. Assume C has q bits and the approximation by far covers its highest r bits. If $r < q$, the bias cannot be neglected after a long time of $O(2^q)$. To address this challenge, we propose to use recirculation to help refresh the values of array periodically and improve accuracy. Egress samples packets with a configurable rate, clones sampled packets and recirculates the copies to ingress with the new queue lengths to replace the stale ones.

3 EVALUATION

We have simulated Tofino switch and implemented Prophet and Prophet supported DRILL application in it. We use real world traces from CAIDA [2] as test traffic, and vary its rate to create different incast ratios. We evaluate Prophet to demonstrate that (1) Prophet could infer queue lengths with high *accuracy*, and (2) Prophet could *effectively support* applications such as DRILL that rely on queue length information in the ingress pipeline.



(a) Inference accuracy



(b) Effectiveness to support DRILL

Figure 2: Prophet inference accuracy & effectiveness

Prophet accuracy. We compare Prophet with two naive solutions: *Infer* that only relies on queue length inference in ingress, and *Recirc* that only uses recirculation to deliver queue length to ingress. Under different incast ratios, we measure the inferred queue lengths produced by the three solutions, and compare them with the accurate queue lengths. The evaluation results are presented in Figure 2(a). We observe that Prophet could produce accurate queue lengths in >90% situations. In comparison, *Infer* could be inaccurate in up to 30% cases, while *Recirc* is inaccurate in up to 60% situations.

Effectiveness of Prophet to support DRILL. Next we demonstrate that Prophet could effectively support applications such as DRILL that rely on queue information in ingress. We configure the incast ratio as 1.2 to create a light congestion situation and see whether DRILL could effectively balance the load based on the queue lengths provided by Prophet. We compare three approaches including DRILL based on accurate queue length, DRILL based on Prophet, and simple ECMP. As shown in Figure 2(b), The average queue lengths of Prophet based DRILL is close to ideal DRILL and is shorter than naive ECMP, and Prophet has a much smaller variation compared with naive ECMP.

4 FUTURE WORK

As our future work, we will extend Prophet to support *advanced queuing methods* such as multi-level feedback queue, PIFO, etc. We will also implement Prophet in *other programmable switches* such as Cavium's Xpliant, Intel's Flexpipe, Cisco's Doppler, Broadcom's Trident 3, etc. to further prove its feasibility and efficiency.

REFERENCES

[1] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *SIGCOMM*.
[2] CAIDA. 2016. The CAIDA Anonymized Internet Traces 2016 Dataset. (2016). https://www.caida.org/data/passive/passive_2016_dataset.xml
[3] Soudeh Ghorbani, Zibin Yang, P Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. DRILL: Micro Load Balancing for Low-latency Data Center Networks. In *SIGCOMM*.