Poster: Scaling Up Anomaly Detection Using In-DRAM Working Set of Active Flows Table

Rhongho Jang[†‡], Seongkwang Moon[†], Youngtae Noh[†], Aziz Mohaisen[‡], and Daehun Nyang[†]
[†]INHA University [‡] University of Central Florida

Abstract—**In the zettabyte era, per-flow measurement becomes more challenging owing to the growth of both traffic volumes and the number of flows. Also, swiftness of detection of anomalies (e.g., DDoS attack, congestion, link failure, and so on) becomes paramount. For fast and accurate anomaly detection, managing an accurate working set of active flows (WSAF) from massive volumes of packet influxes at line rates is a key challenge. WSAF is usually located in a very fast but expensive memory, such as TCAM or SRAM, and thus the number of entries to be stored is quite limited. To cope with the scalability issue of WSAF, we propose to use In-DRAM WSAF with scales, and put a compact data structure called FlowRegulator in front of WSAF to compensate for DRAM's slow access time by substantially reducing massive influxes to WSAF without compromising measurement accuracy. We prototype and evaluated our system in a large scale real-world experiment (connected to monitoring port of our campus main gateway router for 113 hours, and capturing 122.3 million flows). As one key application, FlowRegulator detected heavy hitters with 99.8% accuracy.**

I. Introduction

Due to the high speed network, traffic measurement now have to cope with enormous incoming data rates (*i.e.,* larger number of flows) with tight deadlines (*i.e.,* real-time). We stress that large-scale instant measurement is highly necessary for network anomaly detection. For example, if denial of service (DoS) attacks cause an influx of packets at 100 Gbps, detection delay of 100 ms will cause 1.2GB data to hit a server or a network. Therefore, to avoid large bandwidth penalties, instant anomaly detection is essential.

A working set of active flows (WSAF) is a type of cache of a full flow table, which can be found usually in TCAM (Ternary Content Addressable Memory), CAM, or sometimes SRAM for traffic monitoring. For instanse, NetFlow uses TCAM for storing WSAF in which an entry consists of a flow ID and the counting value. However, the number of entries of the table cannot be large because those types of memories are quite expensive. For scalability, we can put WSAF in DRAM instead of the expensive memory (*i.e.,* incentive to cost-effectiveness). However, there is a speed issue for In-DRAM WSAF: a packet arrival rate is too fast to handle by In-DRAM WSAF, owing to the DRAM's speed and WSAF table's hash collision.

To cope with these issues, sketch-based techniques have been greatly enhanced over several decades because sketch-based counting algorithms only require a small amount of memory to *encode* a large volume of traffic in real-time. Due to their design, however, most of *decode* algorithms involve hundreds of hash calculations and memory accesses from statistically mixed random blocks to obtain meaningful statistics [1]. For this reason, offline decoding in a high-performance server is commonly accepted in practice but inherently incurs huge network delay. Thus, online decoding is highly necessary for instant measurement and further timely detection.

Unfortunately, most sketch-based algorithms lack scalability and online decoding capabilities. Our approach to solve these two problems is 1) to use a counting algorithm that can perform online decoding and 2) to put a flow regulator before WSAF to slow down the incoming packet rate to WSAF. To realize both ideas, we designed a highly scalable counting and flow regulating algorithm called FlowRegulator. By design, instead of directly inserting or updating every packet of a flow into WSAF table, FlowRegulator (*i.e.,* a small cache buffer) retains a fraction of flow counts. By doing so, we can suppress frequent WSAF updates in DRAM; thereby FlowRegulator can support large-scale influx of flows with the use of cost-effective large DRAM. Consequently, FlowRegulator relaxes the necessity of expensive memories (TCAM or SRAM) for maintaining large WSAF, and further enables us to build a highly scalable and fast measurement system. We conducted a real-world campus network experiment for 113 hours by connecting our prototype of FlowRegulator to a mirroring port of a main gateway router, capturing 9.11 billion packets, 122.3 million flows, and 8.5TB bytes. FlowRegulator successfully measured the whole L4 flows with a standard error (0.65%) and detected heavy hitters with 99.8% accuracy.

II. FlowRegulator Design

Our large WSAF in DRAM is in contrast to the small WSAF in TCAM (*i.e.,* industry practice). In DRAM, we can store much more flows, thereby, we do not need a remote collector for decoding. However, the downside is that we cannot evade the "sluggishness" of DRAM.

FlowRegulator to relax the $\{$ips = pps$\}$ constraint: Instead of directly inserting or updating every flow packet into the table, we put a small buffer called FlowRegulator to retain a fraction of flow counts before WSAF. FlowRegulator has a memory block (or a virtual vector initialized to all 0's) for every single flow, and whenever a packet comes in, the corresponding block is updated by setting a random bit of the block. When the block saturates (or a portion of block has set to 1's), the resulting counting fraction (we note that this is not the total size of a flow) is added up to the WSAF (*i.e.,* a hash table in DRAM). Because FlowRegulator retains mice flows whose sizes are lower than the saturation condition, not all the packets are fed into WSAF, but only the packets that trigger

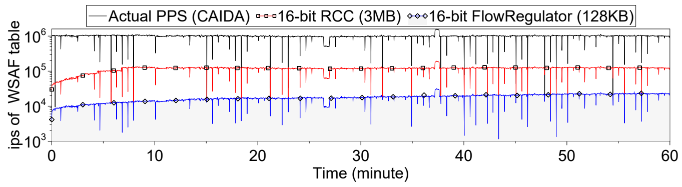


Fig. 1. WASF relaxation: FlowRegulator (FR) and RCC ips of CAIDA dataset

the saturation condition are given to WSAF. This design will greatly reduce insert per second (ips) even under a high packet per second (pps) condition.

To develop FlowRegulator, we utilize the recyclable counter with confinement (RCC) [4] that already has online decoding capability, and proven to be useful for measurement in the wireless SDN environment [2], [3]. To investigate its feasibility, we have tested RCC for its rate regulation (defined as Output ips/Input pps). Given that access time of SRAM is 10-20 times faster than DRAM's (and even faster with TCAM), RCC's rate regulation should be less than 5%. However, its regulation and retention capacity (the maximum number of packets in a virtual vector) are not operationally sufficient. Thus, it is impossible to work with RCC for building FlowRegulator. One way to increase the rate regulation is to give RCC a larger virtual vector, but that does not expand the retention capacity.

Two-layer design for higher rate regulation: Here, our observation is that enlarging the virtual vector size increases the retention capacity just in an addictive manner, and thus, this is not a viable (*i.e.,* scalable) option. Instead, we designed a new counting algorithm for FlowRegulator, which has two layers of probabilistic counters to achieve the higher rate regulation. Our FlowRegulator plays a key role of retaining flows (from feeding into WSAF) for a while as well as counting flows. In the two-layer design, the second (higher) layer's one bit encodes multiple packets of a flow from a saturated sketch of the first (lower) layer. This design has substantially improved the rate regulation in a multiplicative manner. It enables higher rate regulation while not being detrimental to the accuracy and speed, while being scalable.

III. Evaluation

WASF ips relaxation. In Fig. 1, the x-axis represents the time line of our CAIDA dataset, and the black solid line on the top represents the actual pps of the trace. Below the pps line, RCC's and FlowRegulator's regulation rates are shown in red squares and blue diamonds, respectively. The figure shows that RCC relaxes ips to feed packets to WSAF table at the speed of 112 kips (thousand ips), which corresponds to 12% regulation rate. FlowRegulator effectively regulated flows to pass only 1.02% with 128KB DRAM memory. As results, FlowRegulator has sufficient margin, while RCC does not have as can be seen in Fig. 1.

Monitoring in the wild. We implemented our FlowRegulator in an off-the-shelf device and measured up-link traffics (1 Gbps bandwidth) at the backbone gateway (Juniper EX9208 switch) of our campus for 113 hours in total. During 113 hours, 9.1 billion packets of 122.3 billion L4 flows were measured simultaneously both in packets and in bytes.

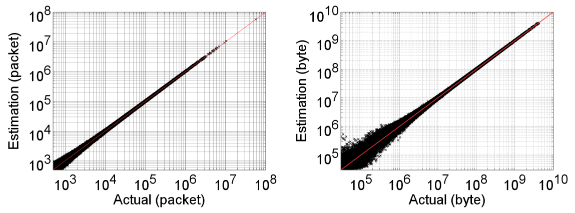


Fig. 2. Accuracy of packet counting (left) and byte counting (right).

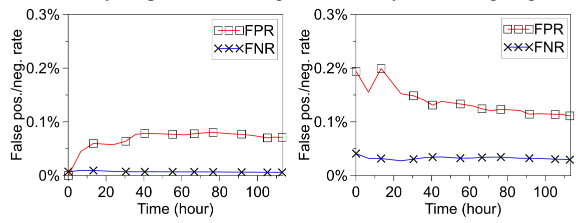


Fig. 3. False positive and false negative rates of packet heavy hitter detection (left) and byte volume heavy hitter detection (right).

Accuracy. FlowRegulator used 128KB for sketch, and 33MB for the WSAF table. Sketches and WSAF table are all in DRAM. Fig. 2 shows the estimation accuracy by standard error for the real-world experiment. For packet counting, we report 0.54% standard error over 350 flows of which size is 1000K+, 1.61% over 11,047 flows for 100K packets, 3.46% over 104292 flows for 10K+ packets. For byte counting, we report 0.63% over 414 flows of which byte size is 1G+, 1.74% over 12,125 flows of 100MB+, 3.65% over 107,726 flows of 10MB+. This accuracy matches the accuracy observed in the lab experiment with the CAIDA dataset.

Heavy hitter detection. Fig. 3 shows FlowRegulator's heavy hitter detection accuracy in terms of false positive/negative rate. Owing to FlowRegulator capability of counting both in packets and in bytes, it can detect both packet heavy hitters and byte heavy hitters. False negative rates in both cases are negligible, and the false positive rates of packet/byte heavy hitters are less than 0.1% and 0.2%, respectively.

IV. Conclusion

In this work, we have developed FlowRegulator for instant flow monitoring. Our approach is different from conventional measurement frameworks by introducing a new notion of very large In-DRAM working set of active flows. In the future work, we plan to demonstrate FlowRegulator's performance and feasibility through an extensive analyses.

Acknowledgement. This work was supported by NRF grant number 2016K1A1A2912757 (Global Research Lab Initiative).

References

[1] Q. Huang, X. Jin, P. P. C. Lee, R. Li, L. Tang, Y. Chen, and G. Zhang. Sketchvisor: Robust network measurement for software packet processing. In *Proc. of ACM SIGCOMM*, pages 113–126, 2017.

[2] R. Jang, D. Cho, A. Mohaisen, Y. Noh, and D. Nyang. Two-level network monitoring and management in WLAN using software-defined networking: poster. In *In Proc. of ACM WiSec*, pages 279–280, 2017.

[3] R. Jang, D. Cho, Y. Noh, and D. Nyang. Rflow+: An sdn-based wlan monitoring and management framework. in Proc. of INFOCOM, 2017.

[4] D. Nyang and D. Shin. Recyclable counter with confinement for real-time per-flow measurement. *IEEE/ACM Trans. Netw.*, PP(99):1–1, 2016.

Poster: Scaling Up Anomaly Detection Using In-DRAM Working Set of Active Flows Table

Rhongho Jang
r.h.jang@knights.ucf.edu

Seongkwang Moon
skmoon@seclab.inha.ac.kr

Youngtae Noh
ytnoh@inha.ac.kr

Aziz Mohaisen
mohaisen@ucf.edu

DaeHun Nyang
nyang@inha.ac.kr

INTRODUCTION

In-DRAM working set of active flows (WASF)

Background Due to the high speed of network, large-scale instant measurement is highly necessary for the network anomaly detection.

Problem For monitoring traffics, NetFlow uses TCAM for storing WSAF in which an entry consists of a flow ID and the counting value. However, the number of entries of the table cannot be large because those types of memories are quite expensive.

Idea For scalability, we can put WSAF in DRAM instead of the expensive memory (i.e., incentive to cost-effectiveness).

Challenges However, there is a speed issue for In-DRAM WSAF: a packet arrival rate is too fast to handle by In-DRAM WSAF, owing to the DRAM's speed and WSAF table's hash collision.

Previous approaches Sketch-based techniques have been used only require a small amount of memory to encode a large volume of traffic in real-time. Unfortunately, most sketch-based algorithms lack ***scalability*** and **online decoding** capabilities.

Our approach

Key idea Put a flow regulator in front of WSAF to compensate for DRAM's slow access time by substantially reducing massive influxes to WSAF.

Relax the {ips = pps} constraint:

- Without FlowRegulator, packet per second (pps) of traffics equal to insert per second (ips) of WASF (*i.e.,* ips = pps).
- With FlowRegulator, we retain a fraction of flow counts before WSAF. When the flow counts saturates, the counting value is added up to the WSAF (*i.e.,* ips < pps).

Features of FlowRegulator:

- Small memory usage (*i.e.,* hundreds of KB) to fit the L1 cache.
- Scalable counting capacity (*i.e.,* retaining portion of large flows).
- Small flow sampling (*i.e.,* probabilistically).
- Online decoding capacity (*i.e.,* extract counting values*)*.

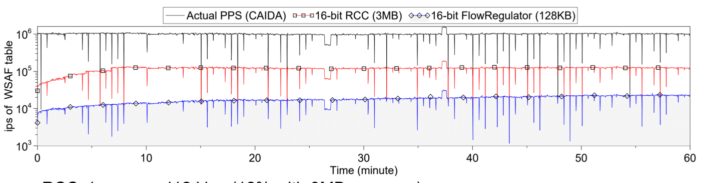
Implementation

- Two-layer design of Recyclable Counter with Confinement (RCC, published in ToN) for scalable ips reduction (*i.e.,* ips << pps).

Evaluation

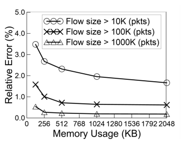
Laboratory experiment (CAIDA dataset)

WASF {pps = ips} relaxation

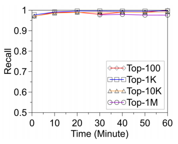


- ***RCC*** 1 mpps →112 kips (12% with 3MB memory).
- ***FlowRegulator*** 1 mpps → 10 kips (1.02% with 128KB memory).

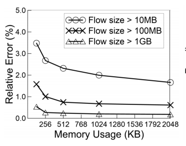
Accuracy (standard error)



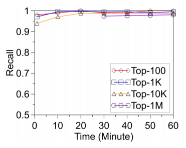
Error rate vs. memory usage



Quality of packet Top-K list



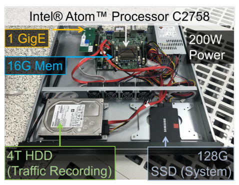
Error rate vs. memory usage



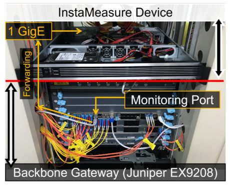
Quality of byte Top-K

- ***Packet counter (256 KB mem.)*** 0.28% (1000K+ flows), 0.99% (100K+ flows),
- ***Byte counter (256KB mem.)*** 0.27% (1GB+ flows), 1.00% (100MB+ flows)
- ***Top-K identification*** Recall > 95% (top 1 million flows).

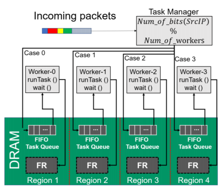
Prototype



Hardware configuration



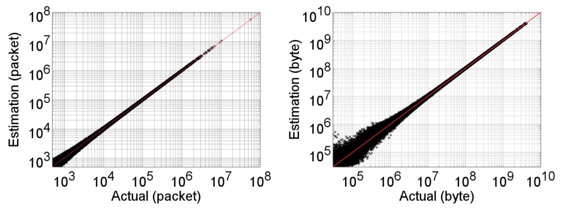
Scenario (campus gateway)


Incoming packets
Task Manager
$Num\_of\_bits(SrcIP) \% Num\_of\_workers$
Case 0
Case 1
Case 2
Case 3
Worker-0 runTask () wait ()
Worker-1 runTask () wait ()
Worker-2 runTask () wait ()
Worker-3 runTask () wait ()
DRAM
FIFO Task Queue
FR
Region 1
Region 2
Region 3
Region 4

Multi-core design (DPDK)

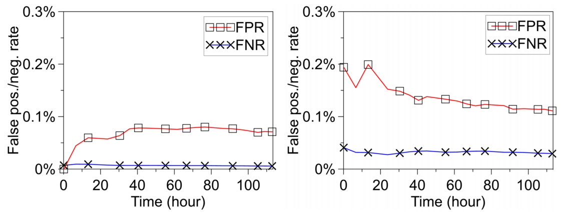
Real-world experiment (113-hour campus gateway)

Accuracy



- ***Packet counter (128KB mem.)*** We report 0.54% standard error over 350 flows of which size is 1000K+,1.61% over 11,047 flows for 100K+, 3.46% over 104,292 flows for 10K+.
- ***Byte counter (128KB mem.)*** We report 0.63% over 414 flows of which byte size is 1G+, 1.74% over 12,125 flows of 100MB+, 3.65% over 107,726 flows of 10MB+.

Performance (Heavy hitter & CPU & MEM)



False positive and false negative rates of packet heavy hitter detection (left) and byte volume heavy hitter detection (right).

- ***Heavy hitter*** False negative rates in both cases are negligible, and the false positive rates of packet/byte heavy hitters are less than 0.1% and 0.2%, respectively.
- ***CPU*** The core's workload matches the traffic pattern, and the core usage did not go over 40% at any point.
- ***Memory*** As for the queue (black diamonds in the figure), it did not grow noticeably.

The results confirmed that *FlowRegulaor* implemented on Atom board worked well for the 1 Gbps network monitoring, and for a quite long time.

NDSS Symposium,
San Diego, California
24-27 February 2019

This work was supported by
NRF grant number 2016K1A1A2912757
(Global Research Lab Initiative)