CoFilter: A High-Performance Switch-Assisted Stateful Packet Filter

Jiamin Cao, Jun Bi, Yu Zhou, Cheng Zhang*
Tsinghua University

CCS CONCEPTS

• Networks → Programmable networks;

KEYWORDS

P4, stateful packet filter

ACM Reference Format:

Jiamin Cao, Jun Bi, Yu Zhou, Cheng Zhang. 2018. CoFilter: A High-Performance Switch-Assisted Stateful Packet Filter. In *Posters-Demos '18: Posters and Demos, August 20–25, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/3234200.3234251>

1 INTRODUCTION

Packet Filtering is a significant mechanism for many network functions, such as firewalls in enterprise networks and isolation in cloud networks. In particular, the Stateful Packet Filter (SPF) is one of the leading Packet Filtering techniques. Through tracking connection states and dropping packets that trigger abnormal states, SPF enables powerful defense against challenging network attacks, such as DNS Reflection Attack and NTP Amplification Attack. Nowadays, SPF is typically implemented in software, *e.g.*, iptables [1]. However, the software-based SPF implementation comes with *significant performance compromise*. For iptables, the throughput is less than 10 Gbps per CPU core, and the delay can be up to several hundred μ s.

Programmable switches and P4 open new opportunities to implement a *high-performance stateful packet filter*. P4

*Jiamin Cao, Jun Bi, Yu Zhou, and Cheng Zhang are with Institute for Network Sciences and Cyberspace, Tsinghua University and Beijing National Research Center for Information Science and Technology (BNRIST). This poster is supported by the National Key R&D Program of China (No.2017YFB0801701) and National Science Foundation of China (No.61472213). Jun Bi (junbi@tsinghua.edu.cn) is the corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Posters-Demos '18, August 20–25, 2018, Budapest, Hungary

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-5915-3/18/08...\$15.00

<https://doi.org/10.1145/3234200.3234251>

Figure 1: Design overview of CoFilter.

provides more flexibility than before to define data plane functions, enabling fast innovations. With high throughput (Tbps) and low latency, programmable switches achieve wire-speed packet processing. However, they have limited memory and somewhat restricted operations, and on their own could hardly support high-performance stateful filtering.

Recent work accelerates SPF based on *switch memorization* [2]. Commercial off-the-shelf (COTS) servers track per-flow states, while switches memorize stateful *packet filter results* as exact match entries and perform stateless operations. Each table entry includes a 104-bit 5-tuple as the match key and some additional action data bits. However, storing ten million table entries takes hundreds of MB SRAM, which is far more than 50-100 MB SRAM available in the latest generation of switching ASICs [4]. Therefore, 5-tuple based switch memorization cannot support massive concurrent flows and is easy to break down under massive attack traffic.

We propose a high-performance, hybrid stateful packet filter, *CoFilter*, which makes the best use of servers and switch ASICs. Different from the 5-tuple based switch memorization, CoFilter exploits a hash-based switch memorization to compress data. The switch uses one default table entry to hash 5-tuples of all flows to store indexes in a register array, and uses some other exact match table entries to map every collided flow to a new index. Hash collision calculation needs considerable memory and calculation capacity, and therefore is implemented in the server. The server maintains information of all ongoing flows, calculates hash collisions and issues instructions to add/delete table entries in switches. Therefore, a copy of the packet that starts or finishes a flow is sent to the server, while other packets are directly processed in switch pipelines. Compared to 5-tuple based switch memorization, which sent copies of all packets that can update states, CoFilter lowers the communication offload between the switch and server.

Figure 2: The table flow graph of CoFilter.

The contributions of this poster are: (1) We propose a hash-based mechanism which can support 40 million flow states in a single switch. (2) We design a hybrid stateful packet filter, CoFilter, which offloads the full stateful strategies to the switch and assigns the server to calculate state storing position in a switch register array, via sending copies of a small number of packets to the server. (3) We implement a prototype on Barefoot Tofino and use ConnTrack [3] as a baseline to identify that CoFilter reduces 40x latency compared with software-based solutions.

2 DESIGN

To reach high throughput and low latency while scaling up to tens of millions of concurrent flows, CoFilter is based on efficient cooperation between the programmable switch and the server, as shown in Figure 1. The switch implements a hash-based pipeline that can efficiently index, store, and track states of substantial flows. Considering the limit memory and capacity of switching ASICs, a backup server calculates hash collisions and notifies the switch via issuing instructions.

Switch: The switch stores states of all connections so that stateful strategies can be performed directly in switch pipelines without getting through the server.

For each packet, the parser extracts $\langle key, state_id \rangle$ from the packet header. *key* defines the flow and can often be the 5-tuple, while *state_id* specifies the accurate state this packet represents. Four types of match-action tables are exploited. Firstly, *Copy_Table* selects packets that start or finish a connection according to the *id* and sends copies of them to the server. Then *Hash_Table* executes a hash function $Flow_ID = hash(key)$. The output *Flow_ID* determines the position in a register array. When two different *keys* are hashed to the same *Flow_ID*, a collision happens. *Collision_Table* stores *key* of collided flows in the match field of table entries and rewrites their *Flow_ID* via executing corresponding actions. In this way, all flows are stored separately. Finally, *Reg_Table* implements state shifting and filtering according to *state_id* using stateful ALUs in registers.

To further eliminate collisions and scale up to more flows, we adapt our design to the multi-pipeline and multi-stage structure of switching ASICs. (1) *Make full use of all pipelines.* Generally, packets arriving at different ingress ports belong

(a) No. of offloaded connections. (b) Latency of CoFilter and OVS.

Figure 3: Offloading capacity and latency test.

to different flows. Therefore, packets are assigned to different pipelines according to ingress ports. (2) *Make full use of all stages.* We use multi-level hash based on the resources allocation in multiple stages of a pipeline, as shown in Figure 2. Packets first come to a *Hash_Table* which maps the 5-tuple to an *n*-bit *Flow_ID* and then move to one of the 2^m MATs according to the first m bits of *Flow_ID*. Shorter *Hash_Tables* and *Collision_Tables* map the remaining $n - m$ bits to an *r*-bit position in a register array. r is determined by the register size in one stage, which is 2^r bits. n should satisfy $r < n < 104$ and we let $n = (r + 104)/2$ in our evaluation.

Server: The server calculates hash collisions and manages *Collision_Table* entries by issuing instructions to the switch. To this end, the server maintains information of all ongoing flows. Luckily, only the existence instead of an accurate state of a flow is required. Therefore, only copies of packets that start or finish a flow need to be sent to the server, e.g., SYN/FIN packets, while other packets are directly processed in switch pipelines.

3 EVALUATION

CoFilter is evaluated in two ways. (1) *We use C language to simulate CoFilter to test offloading capacity.* We use trace from MAWI [5] as test traffic. Figure 3(a) shows the max flows CoFilter can offload. With available register size rising, the switch pipeline is always able to offload more flows. But further increase in the size of *Collision_Tables* has little effect, as a small one is enough to store collided flows. When the register size comes to 2^{22} bits, which is the upper limit in one stage, the switch pipeline can offload more than 45M (10^6) flows. (2) *We build CoFilter with a Tofino switch and a backup server to test latency.* In our experiment setup, we use another two servers, one for the OVS ConnTrack as a contrast, and the other acting as traffic generator and receiver for latency test, each of which has an Intel XL710 40G NIC. We perform comparisons of latency for various packet sizes, to compare the performance of a native software-based implementation of ConnTrack, against our switch-assisted design. As is shown in Figure 3(b), the software solution incurs more than 40μs latency on average and large variance, while the latency of CoFilter is 1μs and 40x lower than that of software-based, with much smaller variance.

REFERENCES

- [1] Harald Welte, Pablo Neira Ayuso. 2014. netfilter/iptables. Website. (2014). <https://www.netfilter.org/>.
- [2] Georgios P Katsikas and et al. 2018. Metron: NFV Service Chains at the True Speed of the Underlying Hardware. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. USENIX.
- [3] Linux Foundation. 2018. OVS Conntrack Tutorial. Website. (2018). <http://docs.openvswitch.org/en/latest/tutorials/ovs-conntrack/>.
- [4] Rui Miao and et al. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 15–28.
- [5] WIDE Project. 2018. MAWI Working Group Traffic Archive. Website. (2018). <http://mawi.wide.ad.jp/mawi/>.