Poster: CO2: Collaborative Packet Classification for Network Functions with Overselection

Yunhong Xu[*], Hao Wu[†], Nick Duffield[*], Bin Liu[†], Minlan Yu[‡]
[*]Texas A&M University, [†]Tsinghua University, [‡]Harvard University

***Abstract*—The growing number of network functions drives the need to install increasing numbers of fine-grained packet classification rules in the network switches. However, this demand for rules is outstripping the size of switch memory. While much work has focused on compressing classification rules, most of this work proposes solutions operating in the memory of a single switch. This paper proposed, instead, a collaborative approach encompassing switches and network functions: we couple approximate classification at switches with fine-grained filtering where needed at network functions to accomplish overall classification. This architecture enables a trade-off between usage of (expensive) switch memory and (cheaper) downstream network bandwidth and network function resources. Our implementation uses approximate classification and Prefiltering to reduce switch memory usage. Our system can reduce memory significantly compared to a strawman approach, as shown by simulations of real traffic traces and rules.**

I. Introduction

Software Defined Networking (SDN) is a powerful enabler for Network Function Virtualization (NFV). By moving network appliance functionality from proprietary hardware to software, NFV promises to bring greater openness and agility to network data planes [1]. An SDN controller can install classification rules at switches and set up tunnels to direct the selected traffic to receivers that run various network functions [2]. In the cloud context, traffic is classified for different L4-L7 network functions such as software load balancers [3], WAN optimizers, and proxies. Operators may mirror packets to analysis functions (e.g., intrusion detection, or deep packet inspection) to debug network problems [4].

In enterprise networks, about half of network devices are middleboxes that provide network functions [5]. These functions can require millions of fine-grained traffic rules. Tens of thousands of 5-tuple flows may be active at each top-of-rack switch rack [2], [6], each potentially requiring a rule to mirror selected packets towards network analysis functions [4].

The current switch memory is insufficient to store these numbers of rules. Even storing 500K 5-tuple prefix rules exhausts on-chip SRAM (e.g., $20-60MB$ in Trident2 [7])), most of which is already used for regular packet forwarding and other match-action rules [8]. The reduction in memory usage attainable by rule compression [9]–[11] does not change the conclusion, with compression ratios of e.g. 29% for 2 decisions and 48.3% for more distinct decisions classifier [11]. While approximate data structures such as Bloom filters and Cuckoo filters [12], can reduce the memory usage by a further 50%, no systematic treatment of the consequences of the resulting false positives has been provided.

This paper addresses the problem of how to store and apply large numbers of classification rules to network traffic. Our work is based on the observation that network bandwidth and downstream processing resources are relatively cheap compared with switch memory. We propose *COllaborative Overselection (CO2)* , a framework for packet classification that distributes classification functionality between switches and the network functions at end hosts in order to achieve the best trade-off between switch memory, communication bandwidth, and computational resources. Figure 1 shows the CO2 architecture. At a high level, the SDN controller installs the traffic classification rules at the switch. The switch performs Approximate Packet Classifier (APC) based on these rules, which has the side-effect of forwarding the overselected packets to the receivers. The receivers detect overselection and provide feedback to the controller, which then dynamically reconfigures the classifiers at the switches in order to control the amount of overselection.

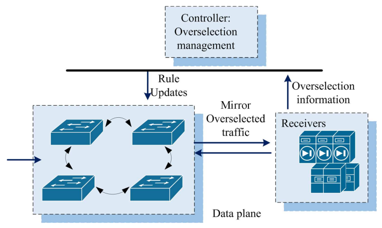
Controller: Overselection management
Rule Updates
Overselection information
Mirror Overselected traffic
Receivers
Data plane

Fig. 1. CO2 System Architecture.

We summarize the contributions and provide the paper outline as follows,

1) *Reduction in Switch Memory Usage.* Section II details switch design of approximate prefix matching. A *Prefilter*, placed in front of the APC, provides a transparent mechanism to intercept the *overselected* traffic.
2) In Section III, we evaluate the performance of CO2 under the configurations of real network traces.

We conclude and describe the future work in Section IV.

II. CO2 Switch Design

In this section, we describe the detailed operation of the components of the CO2 and their capabilities used to trade off the memory usage and the downstream bandwidth. Section II-A describes the *Prefilter* to reduce the *overselection*

Annex to ISBN 978-3-903176-28-7©2020 IFIP.

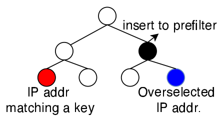

Fig. 2. An example of a *Blackkey*.

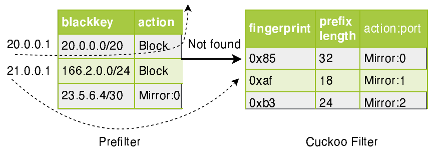

Fig. 3. A *Prefilter* and Cuckoo filter for 1-field rule set.

rate, where *overselection rate* is a term defined by the *overselected* traffic in bytes divided by the wanted traffic required by the rule set. The information concerning overselection detected at the receivers and how it can be used to adapt the switch configuration to maintain conformance to overselection targets are described in Section II-B.

A. Overselection Reduction

We describe approximate prefix matching of key prefixes using Cuckoo filters [12] in the APC. Cuckoo filter storage locations are managed, in part, by a hash of the input known as the *fingerprint*. The Cuckoo filter as well adapted to the CO2 architectural principles because the field length of the fingerprint determines the hash collision rate and hence the *overselection rate*.

One strawman approach of storing a large number of rules into a switch is to use a Cuckoo filter whose entry is the fingerprint generated by hashing a rule, which we call it *Cuckoo Only* approach. However, if we have a small memory available for the rules, the only thing we can do with the base approach is to reduce the fingerprint size, but this will cause a large number of wrong selections.

We propose to employ Prefiltering to reduce the overselection by intercepting the traffic (both wanted and unwanted) before it reaches the APC. In order to optimize the use of memory in the switch, the *Prefilter* is configured to treat those flows responsible for the largest overselection, as determined by the receivers and the controller. The *Prefilter* may employ any data structure that supports exact membership queries. We implement the *Prefilter* with a Cuckoo hash table to ensure constant-time lookup for incoming packets.

We describe the two operations of the *Prefilter* under this configuration. First, the *Prefilter* may blacklist designated unwanted traffic by blocking packets of selected IP prefixes; we call these *Blackkeys*. One *Blackkey* is generated by finding the highest ancestor of the *overselected* IP address until a key is found. The purpose of this is to block all the traffic which matches this ancestor. Figure 2 shows an example of the *Blackkey*, where the red node denotes an IP address that matches a rule prefix, and the blue node indicates an IP address that does not match any rules. Instead of putting the blue node into a *Prefilter*, we obtain the highest ancestor (the black node in Figure 2), which does not include any rule prefixes. Second, the *Prefilter* whitelists a subset of the original rules by applying their stated actions to matching traffic; we call the corresponding rule prefixes *Whitekeys*. Whitelisting may be used to map to the correct action for a large flow, which is subject to hash-based overselection.

Combining the *Prefilter* with Cuckoo filter, the procedure of the selection pipeline is as follows: when a packet arrives at the switch, it first checks membership with the *Prefilter*; if no match is found, the packet consults the Cuckoo filter for further actions. One example of the pipeline is shown in Figure 3. The *Prefilter* includes 3 rules, two of which are blackeys to block, and one is a *Whitekey* to port 0. The Cuckoo filter compromises 3 rules, and each includes a fingerprint, a prefix length, and an action. The packet with IP address 20.0.0.1 matches the first entry in the *Prefilter* and is blocked by *Prefilter*. Another packet with IP address 21.0.0.1 obtains no membership in *Prefilter*, then matches the second entry of the Cuckoo filter, and is mirrored to port 1. Note that this pipeline is for measurement and does not influence the routing path of the packets.

B. Prefilter Update

Overselected packets are identified by the absence of a map entry for them. The receiver maintains a list of byte volumes of *overselected* IP prefixes, either exactly or estimated using an approximating data structure such as a Count-Min sketch [13]. Overselection notifications are then generated periodically for prefixes that exceed the receiver's configured overselection target. Additionally, to reduce overhead in computation and transmission, receivers only transmit some proportion of these notifications in decreasing order of overselection size.

The *Prefilter* is dynamically configured by the controller on the basis of information supplied by the receivers. The *Prefilter* updating is similar to an LRU cache [14], where the oldest items in the *Prefilter* will be evicted if the capacity is exceeded. The goal is to install both the biggest IP addresses/prefixes or IP addresses/prefixes with the highest overselections to the *Prefilter*. Once the controller gets the feedbacks, it identifies the associated *Blackkeys* or *Whitekeys*. Those keys will be installed to the *Prefilter* in decreasing order of *overselected* traffic volume, up to the *Prefilter* capacity. Entries remaining in the *Prefilter* from the previous period are evicted by time counts to accommodate feedback IP addresses/prefixes from the current period.

III. Evaluation

In this section, we evaluate the performance of CO2 under the configurations using realistic rules and traffic traces. Our simulation results show that at a 1% *overselection rate*, CO2 uses 23.66% less memory than the base approach.

A. Experiment setting

Classification rules. We generate a 500k-rule set on source and destination pairs (Prefix pairs) using ClassBench [15]. The set contains 500K distinct source prefixes and 152 distinct destination prefixes, which covers 54 million distinct IP pairs.

Traffic traces. We employ a 1-hour CAIDA trace [16], which contains anonymized passive traffic traces from CAIDA's passive monitors in 2015. The trace contains 32 million distinct IP pairs. For each pair, we randomly map it to one of the IP pairs in the ClassBench rules. After the mapping, all the rules are

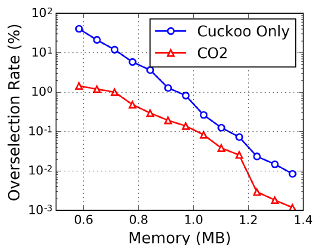

Fig. 4. Comparison between CO2 and *Cuckoo only* in *overselection rate*.

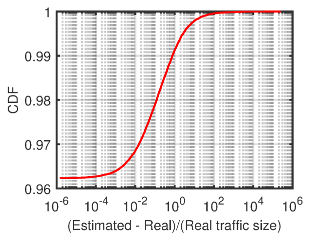

Fig. 5. The CDF plot of the relative byte error per rule.

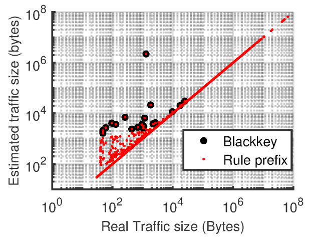

Fig. 6. The scatter plot of estimated vs. real traffic size.

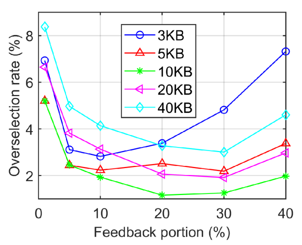

Fig. 7. The *overselection rates* with different feedback portions and *Prefilter* capacities.

ensured to have matching traffic. We mix the mapped traffic with the original trace in an order by the timestamp, which includes traffic that does not match any rules.

B. CO2 performance

We compare the *overselection rate* for CO2 and *Cuckoo only*. The simulation is conducted under fixed memory usages. The result is shown in Figure 4. We see that CO2 achieves a lower *overselection rate* compared to *Cuckoo only*, which indicates that, with the same *overselection rate*, CO2 is more memory saving. It is referred that CO2 uses 23.66% less memory than *Cuckoo only*.

Measurement functions typically report statistics at a finer level of granularity. Therefore we drill down within our previous results to report overselection at a per rule granularity when the memory capacity is $0.71MB$. We found the statistical effects of overselection to be quite limited: 97.0% of rules have a relative byte error of 1% or less, as referred from Figure 5. Figure 6 shows a scatter plot over rules of measured vs. real bytes in a $100ms$-interval and shows that more significant errors are mostly confined to rules with smaller byte volume.

C. Prefilter Size and Feedback Portion

Our results show that a small *Prefilter* of $10-20KB$ suffices, with a feedback rate of $10\%-20\%$. The simulations are conducted to study the dependence of the overselection rate on the capacity of *Prefilter* and the feedback proportions of possible overselection notifications that the receivers send to the controller. From a total switch memory of 0.71MB, we explore *Prefilter* capacities of 3KB, 5KB, 10KB, 15KB, and 20KB, together with feedback proportions of $1\%-40\%$. The results are shown in Figure 7. From it, we observe the *overselection rate* is smallest for feedback ratios of $20\%-30\%$. The reason why the overselection rate begins to increase at 25% feedback is that a larger feedback ratio makes the entries existed in the *Prefilter* evicted to install new Blackkeys/Whitekeys, and this eviction results in a number of large flows (still live) bypassing *Prefilter* to Cuckoo filter. From Figure 7, we can also see that the best *Prefilter* sizes are $10-20KB$ because employing a larger *Prefilter* reduces the length of the fingerprint, which causes more overselections of Cuckoo filter.

Acknowledgment

This work was supported in part by the National Science Foundation under award CNS-1618030, the National Natural Science Foundation of China (68172213,61432009) and by Google.

IV. Conclusion

With the growing number of classification rules, it becomes critical to limit memory usage. CO2 allows approximate classification at switches to reduce memory usage by introducing a small amount of overselected traffic. Our simulations with real traffic traces and rules show that CO2 achieves 23.66% less memory than *Cuckoo only* at the overselection rate of 1%.

We have a few discussions beyond the scope of the work. one concern is the overhead introduced by prefiltering, e.g., CPU usage, delay, and etc. We will study the performance degradation caused by this overhead, especially, in extreme situations, e.g. high network load. Another concern is the trade-off between sizes of prefilter and cuckoo filters. In the future, we may need to find a sweet point which provides optimized trade-off both theoretically and experimentally.

References

[1] S. Palkar and e. Lan, "E2: a framework for nfv applications," in *SOSP*. ACM, 2015.
[2] Z. A. Qazi and e. Tu, "Simple-fying middlebox policy enforcement using sdn," vol. 43. ACM, 2013.
[3] P. Patel, D. Bansal, and etc., "Ananta: Cloud Scale Load Balancing," in *SIGCOMM*, 2013.
[4] Y. Zhu and N. K. etc., "Packet-level telemetry in large datacenter networks," in *SIGCOMM*, 2015.
[5] J. Sherry and e. Hasan, "Making middleboxes someone else's problem: network processing as a cloud service," *SIGCOMM*, vol. 42, no. 4, 2012.
[6] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *SIGCOMM*, 2015.
[7] Ipspace, "Trident 2 Chipset and Nexus 9500." [Online]. Available: http://blog.ipspace.net/2014/06/trident-2-chipset-and-nexus-9500.html
[8] P. Bosshart, D. Daly, and etc., "P4: Programming protocol-independent packet processors," *SIGCOMM*, 2014.
[9] Q. Dong, S. Banerjee, J. Wang, and D. Agrawal, "Wire speed packet classification without tcams: A few more registers (and a bit of logic) are enough," *SIGMETRICS Perform.*, 2007.
[10] O. Rottenstreich and J. Tapolcai, "Lossy compression of packet classifiers," in *ANCS*, May 2015.
[11] A. X. Liu, C. R. Meiners, and E. Torng, "Tcam razor: A systematic approach towards minimizing packet classifiers in tcams," *IEEE/ACM Trans. Netw.*, 2010.
[12] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," ser. CoNEXT, 2014.
[13] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, 2005.
[14] T. R. Puzak, "Analysis of cache replacement-algorithms," 1985.
[15] D. E. Taylor and J. S. Turner, "ClassBench: A Packet Classification Benchmark," *Transactions on Networking*, vol. 15, no. 3, 2007.
[16] "CAIDA Anonymized Internet Traces 2012," http://www.caida.org/data/passive/passive_2012_dataset.xml.