Dear OVSCON committee,

We are writing you to propose an abstract for a full talk at the 5th Open vSwitch Fall Conference 2018 held on Dec 5 and 6 at Club Auto Sport in San Jose, California.

Please find the required details below:

- Title: The Discrepancy of the Megaflow Cache in OVS
- Names, email address, and affiliation for each speaker:

  - **Levente Csikor** (csikor@tmit.bme.hu)
    Research Associate
    High Speed Networks Lab, Budapest University of Technology and Economics, Hungary
  - **Gábor Rétvári** (retvari@tmit.bme.hu)
    Senior Research Associate
    High Speed Networks Lab, Budapest University of Technology and Economics, Hungary

- Type of talk: **Full talk**

1 Background

In the beginning, the aim of Open vSwitch (OVS) was to meet the needs of the open source networking community to provide a feature-rich OpenFlow (OF) software switch offering a sophisticated way to manage the underlying network of virtual machines (VMs) in Linux-based hypervisors (e.g., KVM, XEN). Due to its continuous evolution (just to mention a few: supporting standard management interfaces like NetFlow, complying with the latest OF standards, etc.), it has become the de-facto standard OF software switch implementation, recently has become the most popular network back-end for OpenStack deployments [1], and started to gain a foothold in serverless environments (e.g., OVN/Kubernetes[1]). Although, to catch up with the industrial needs, OVS had to be as fast as possible without sacrificing its most important advantages: high programmability and generality.

To reach this end, two important parts have been introduced to the OVS forwarding plane: the "slow-path" daemon called *ovs-vswitchd* running in the user space, and the "fast-path" flow caching architecture running in the kernel space. In particular, the slow-path materializes the *flow table*, which is an ordered set of wildcard rules operating over certain header fields, and a set of packet processing primitives (i.e., actions) to be applied to matching packets. For increased flexibility, OVS permits flow rules to overlap, however this makes packet classification rather complex since, even in the case of non-overlapping flow rules, the complexity of any wildcard rule matching algorithm could be exponential [2]. On the other hand, to fasten packet classification in OVS, only the first packet of each flow is subjected to this full blown flow-table processing on the slow path, and the flow-specific rules and actions are then cached in the fast path, which can then process the rest of the flow's packets efficiently.

Since OVS 1.11, this fast path is further divided into two layers of *flow caches*: the *microflow cache* that implements a per-transport-connection exact-match store over all header fields; and the *megaflow cache (MF)*, which, by introducing arbitrary bitwise wildcarding to the kernel space, has enabled the biggest performance improvement so far [1]. For the rest of our discussion, we will concentrate on the megaflow cache exclusively.

[1]`https://github.com/openvswitch/ovn-kubernetes`

2 Contribution

Since 2015, we have been investigating the highest attainable performance of OVS in many diverse use cases ranging from pure L2 forwarding to more complex telco access gateway scenarios. In some cases, we have identified some (at that time considered as) inexplicable performance drops with certain flow table entries and traffic traces even if in two measurements the use case itself was the same so did the number of flow table entries and the number of incoming flows. Then, we started to deeply analyze how the flow caching architecture of OVS works and what strategy it uses to build its data structure.

The data structure used in the MF is based on the tuple-space search (TSS) scheme: a tuple is defined for each combination of field length, and the resulted set is the "tuple space". In case of the MF, TSS means that entries matching on the same header fields are collected into a hash in which masked packet headers can be found fast. To reduce complexity, the slow path ensures that MF entries are non-overlapping, resulting in masks and associated hashes are searched sequentially until the first matching entry is found. This means that even if hash lookup is $O(1)$, the TSS algorithm still has to iterate through all hashes assigned to different masks, rendering TSS a costly linear search when there are lots of masks. To understand how this looks like in the reality, we considered a simple Access Control List: we have a flow table with *two overlapping* flow rules; packets coming from the IP range 10.0.0.0/8 are allowed, and everything else should be dropped.

ip_src	action
00001010 ******** ******** ********	allow
******** ******** ******** ********	deny

(a) Binary ACL representation of the single-field network policy

Key	Mask	Action
00001010	11111111	allow
10000000	10000000	deny
01000000	11000000	deny
00100000	11100000	deny
00010000	11110000	deny
00000000	11111000	deny
00001100	11111100	deny
00001000	11111110	deny
00001011	11111111	deny

(b) Resultant non-overlapping MF entries

Figure 1: Simple ACL and the corresponding MF cache.

The corresponding flow table shown in Fig. 1a needs to be materialized in the MF cache in a non-overlapping form. To reach this end, OVS tries to wildcard as many bits as possible to get the broadest possible rules and also reduce the required number of entries per hash tables. This strategy results in an exact-match entry for the allow-rule and 8 different key-mask pairs for testing the rest of the header bits (see Fig. 1b). Note that here we concentrated only on the relevant first 8 bits and omitted the remaining fully wildcarded 24 bits. Naturally, in order to get the above-mentioned realization in the OVS data path a corresponding packet sequence is also necessary to populate the MF cache in such a way. In essence, it means that if all incoming packets are from the 10.0.0.0/8 IP range (meaning 2^{24} possible different IP packets), then there will only be one entry in the MF cache, which the matching can be found blazingly fast for. However, if those 8 specific packets outside of the 10.0.0.0/8 IP range are being caught by OVS (128.0/8, 64.0/8, 32.0/8, etc.), then the MF cache will end up in the above-mentioned state. Of course, even if we match on an exact /32 IP address instead of the whole /8 IP range, iterating through 32 different masks/hash tables would not cause any substantial overhead for OVS[2], yet, if we also define an ACL on another header field, say the L4 destination port, the resulting MF cache could have $32 * 16 = 512$ masks/hash tables (due to the cross product), which could introduce significantly increased computation time until the matching entry is found.

To put these thoughts into numbers, we have carried out extensive performance measurements in

[2]Especially, if OVS has sufficient resources, e.g., many CPU cores.

accordance with this finding (first, in a synthetic KVM-based environment [3]), and it turned out that the tediously designed flow caching architecture offering a ludicrous speed for OVS can, in certain cases, produce the contrary: with certain flow rules and traffic patterns it can be exhausted in such a way that most of the packet classification will be done in the slow path resulting in a huge performance penalty. In particular, we found the even with 512 masks/hash tables[3] we can slow OVS down to $10 - 20\%$ of its peak performance. Moreover, if we also introduce a filtering rule on the L4 source port, it would yield 8192 masks/hash tables resulting in a performance drop to *zero*. Note that due to the fact that an entry (without a subsequent hit) remains "hot" in the MF cache for 10 seconds, hence the sending rate of those 8192 packets does not even need to be fast-paced: rate of less than 1000 packets/s is sufficient. This renders such a use case a covert denial-of-service attack.

During our talk, we will also carry out a live demonstration, where we will show how this technique works in our synthetic KVM-based environment, where a simple *iperf* traffic between two VMs will be exposed to the effects our MF cache exhaustion. We will also discuss that the restrictive access of tenants to the underlying hypervisor switches in production cloud environments (e.g., OpenStack, Kubernetes) is also sufficient to reproduce our finding revealing a possible denial-of-attack surface[4]. Furthermore, we will also present some immediate actions to overcome this problem, e.g., increasing the number of CPU cores available for OVS, disabling MF cache, offloading OVS datapath to a SmartNIC, using jumbo frames (whenever it is possible) to drastically reduce the per-packet load needed to be processed by OVS.

References

[1] J. Pettit, "Accelerating Open vSwitch to "Ludicrous Speed"," Blog post: Network Heresy - Talses of the network reformation, https://networkheresy.com/2014/11/13/accelerating-open-vswitch-to-ludicrous-speed/, 2014.

[2] P. Gupta and N. McKeown, "Algorithms for packet classification," *Netwrk. Mag. of Global Internetwkg.*, vol. 15, no. 2, pp. 24–32, 2001.

[3] L. Csikor, C. Rothenberg, D. P. Pezaros, S. Schmid, L. Toka, and G. Rétvári, "Policy injection: A cloud dataplane dos attack," in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, ser. SIGCOMM '18. New York, NY, USA: ACM, 2018, pp. 147–149. [Online]. Available: http://doi.acm.org/10.1145/3234200.3234250

[3] Note again that it means 512 packets classified by 3 simple flow rules.

[4] Note that we have recently contacted the OVS developers about this attack surface and explained all steps to reproduce such an attack