Where turbo boosting TC flower control path had led us to

Marcelo Ricardo Leitner, Vlad Buslov
Red Hat, Nvidia
mleitner@redhat.com, vladbu@nvidia.com

Abstract

While a lot of effort is put in the community and industry in general to improve datapath performance, the control path does not get as much attention. In this talk we will describe our journey to improve the TC filter control path performance and in particular the control for hardware offloading of Flower rules.

The ability to create, read, update and delete TC filters that are offloaded in hardware is fundamental in getting efficient datapath packet processing in particular if the hardware is used at runtime to cache flows. There are many use cases (e.g. short lived container workloads, telco environments, etc) where the rate of churn of rules is high therefore the faster one can update the datapath rules the sooner the datapath can take effect. Efficient statistics are also critical for some use cases. In billing, for example, the sooner the controlling application knows about it, the smaller and more predictable the over usage is.

In this talk we describe the different improvements we made as well as present results for each enhancement we made over several kernel iterations and the impacts they had on OvS learning rate with HWOL enabled.

Open vSwitch TC offload overview

The general approach to implementing OvS data path with TC is described in detail in [19]. For us the main point of interest is OvS threading model and its mapping to TC messages. Open vSwitch data path is serviced by two sets of threads:

Handlers - receive miss-packets from OvS kernel module and install new flows to TC according to configured OpenFlow rules. Send RTM_NEWTFILTER TC messages to add new filter[1].

Revalidators - validate and delete existing flows from TC when the configuration is changed or flow is aged out due to inactivity [18]. Send RTM_GETTFILTER with NLM_F_DUMP modifier to get statistics for all flows, delete flows by sending RTM_DELTFILTER message.

Multiple handlers and revalidators can send TC netlink messages concurrently, as shown in Fig. 1.

[1]For a deep-dive into Linux Traffic Control Classifier-Action subsystem please consult [20]

Initial TC performance evaluation

In order to be a well-performing data path for Open vSwitch TC control path needs to be scalable (adding millionth flow should not take significantly more execution time than the tenth), allow significant parallelism (to not block concurrent handler and revalidator threads), and not impose additional overhead (TC is just a small part of OvS data path, needs to accommodate user land OvS process on top of the stack and driver/hardware at the bottom). In our initial evaluation, 4 years ago, we found TC severely lacking in all of these which made TC data path unusable for anything besides simple L2 deployments with the maximum number of concurrent flows in low tens of thousands.

To help make the case, lets exemplify it. Because dumping the flows took the same lock (rtnl lock) as adding or removing a filter, during the moments that the revalidators were dumping it, no filter could be added by a handler thread. That also means that the handler thread was blocked waiting for the lock and unable to handle further miss packets. In simpler words, dumping stats led to packet drops, depending on the conditions of the system.

Even with 16 CPUs available to the host, OvS could not learn more than a thousand flows a second after a few thousands of flows, as shown in Fig. 5.

Scalability

TC classifiers and actions had significant bottlenecks that impacted performance on higher scales. Inside a flower classifier instance, individual filters were looked up with linear search over a linked list. Action instances were looked up with a static hash table with few buckets which quickly degraded to linear search on high scale. Moreover, if a user didn't specify id when creating new filter or action (which is the case for OvS since it identifies the flow by the cookie value, ignoring the id), then linear lookup over all range of available ids was performed to find a free slot. This resulted in a quadratic algorithm complexity for both filter/action initialization [21] and dump.

Parallelism

All of the TC classifier API message handlers were synchronized by one-big-lock rtnl_lock. To make things worse, the same lock was used by several other unrelated networking

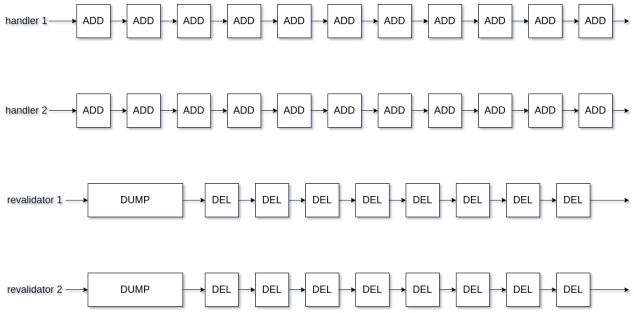

Figure 1: Idealized presentation of execution of concurrent handler and revalidator threads

subsystems which made the lock even more contended in some circumstances.

Performance

Several significant performance bottlenecks become immediately obvious when trying to run OvS with the number of flows in hundreds of thousands:

- Percpu allocator used to allocate action counters consumed tens of percent of CPU time spent in TC, which regressed to 50% CPU time spent in percpu allocation code in kernel 5.0 [10]. The regression was reported and fixed as a bug [23], but even with the fix, we were not satisfied with percpu allocator performance. Note that percpu allocator in our use-case also exhibits problems with scalability (linear search in a linked list of chunks) and parallelism (performs the search with a global lock taken) but since it is an external dependency for TC we consider it a general TC performance issue.
- Filter dump was too slow, even with quadratic behavior fixed. The approach of copying so much data via netlink packets from kernel to user space resulted in significant overhead.
- Low-level performance issues with ConnectX-5 steering implementation. The original approach to managing steering resources was to use the firmware and present an unified API to driver abstracting the differences in implementation details between different hardware generations. However, since the embedded CPU used to run firmware is significantly underpowered compared to the host, it was easily overwhelmed by the high flow update rate. The problem couldn't be remedied by optimizations in firmware and required rethinking our approach to the way we program steering in ConnectX NICs.

Overall, we wanted OvS TC data path to be as close as possible to the idealized representation from Fig. 1, but instead, we observed several bottlenecks and heavy lock contention as shown in Fig. 2.

Tests performed in this paper

The idea is to demonstrate how impactful the TC changes mentioned in this paper were for OvS and how it is today. For that, we tested each major kernel version on which these

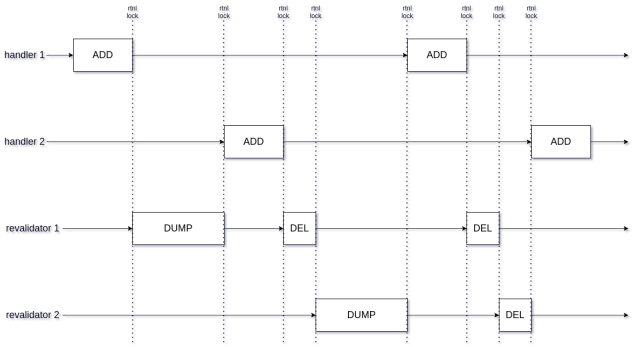

Figure 2: Actual concurrent handlers and revalidators executing with high contention of rtnl_lock before optimizations

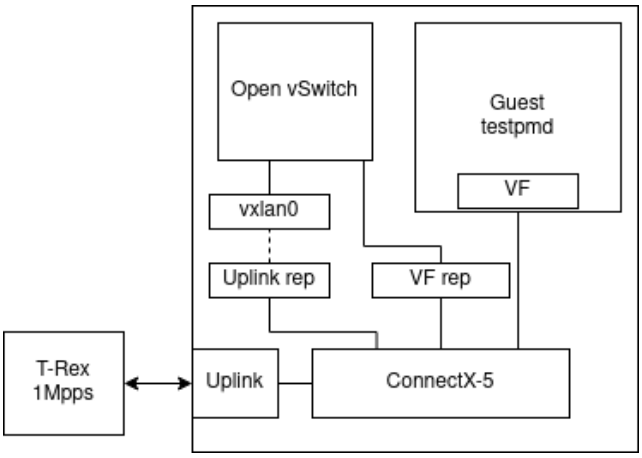

Figure 3: Test topology. Arrows inside the DUT were omitted for brevity.

went in on the DUT, as shown in Fig. 3. The Linux kernel has a lot of changes in between, some more related to the matter than others, and the test results can be affected by it. Notwithstanding, testing such kernels only helps to show the current state of OvS performance, and track the performance trend over time.

We used small UDP packets encapsulated with VXLAN. Small UDP packets minimizes any effort on copying the packet itself, while using the tunnelling stresses a complex part of the control path. As traffic generator, we used T-Rex with a single port which generated packets as fast as 1Mpps. This rate is fast enough to repeat packets often in case of drops while not overwhelming the system with raw packet switching.

In order to get OvS to install many tc rules, we used a varying source MAC address for the inner packet. With that and the following simple OpenFlow flows, OvS will install one tc flow for each source MAC address.

- `in_port="enp130s0f0np0_0" actions=output:vxlan0`
- `in_port=vxlan0 actions=output:"enp130s0f0np0_0"`

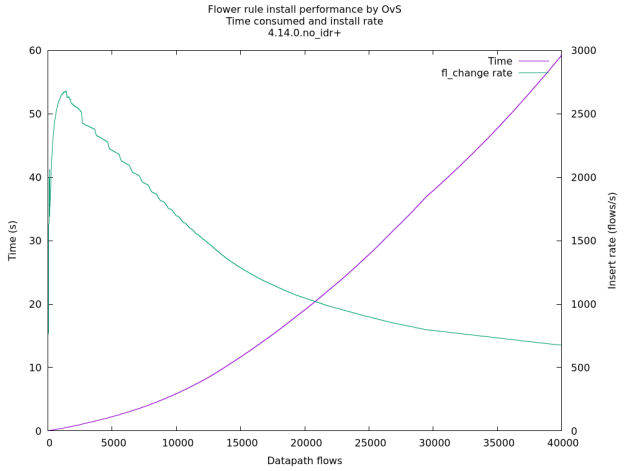

Figure 4: Kernel 4.14 without idr patches and with 4 CPUs available to the host.

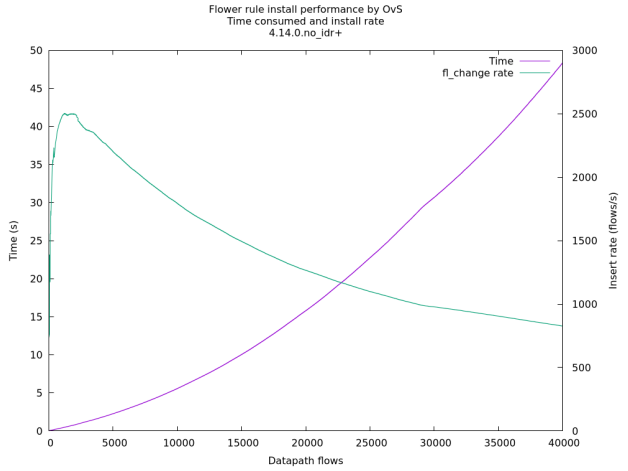

Figure 5: Kernel 4.14 without idr patches and with 16 CPUs available to the host.

- `dl_src=de:ad:00:00:00:00 actions=drop`
- `dl_dst=de:ad:00:00:00:00 actions=drop`

The VF (Virtual Function) is attached to a guest, that runs testpmd in macswap mode. As a result, for each source MAC address used, two TC rules get installed. One on the uplink port and another on the VF representor. Because of the changes that went in on 5.4 we had to increase the number of flows being tested, otherwise the test would become too short and 400K datapath flows is closer to some use cases we know of.

As for hardware, we used a Mellanox ConnectX-5 MT27800 with firmware 16.27.2008, linked at 100Gbps, for both, DUT and T-Rex. The host had a Intel Xeon E5-2690 v4 @ 2.60GHz with tuned configured to cpu-partitioning profile, isolating the guest and CPU cores that we wouldn't use. As the NIC was plugged in NUMA 1, we isolated all NUMA 0 cores, and varied the amount of free NUMA 1 CPUs for the host in some tests. Although the guest had 12 CPUs allocated to it, testpmd used only 1.

The DUT was running RHEL 8 user space with Red Hat's package of Open vSwitch 2.15, which was synced to upstream 2.15 stable branch on Jun 11th and has support for terse dumps and to use TC's new no_percpu.

In the graph results of this paper the time line is read against the left y axis and represents the cumulative test time, time that is needed to insert that many datapath flows. The fl_change rate line is read against the right y axis and represents the cumulative average of insertions done throughout the test.

Long journey behind

Solving obvious scalability issues

The scalability issues described in the Initial TC performance Evaluation section are programmer's dreams as they are textbook examples of usage of linear algorithms that can be trivially substituted with optimized data structures. For both

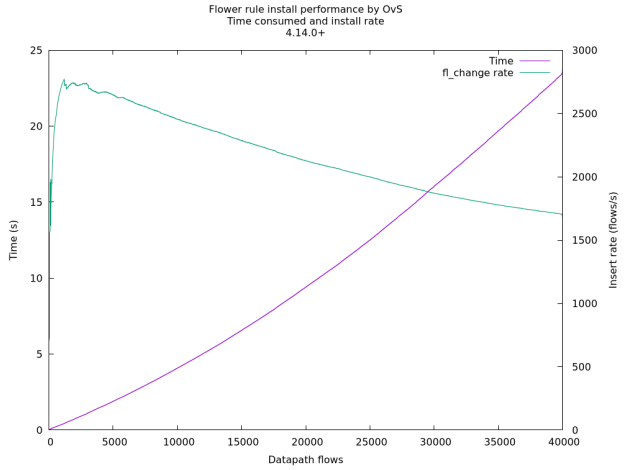

Figure 6: Kernel 4.14 with 16 CPUs available to the host.

flower classifier and action API the data structure of choice was idr (radix tree), which solves the need for id allocation in these modules and provides optimized lookup. Indeed, with only 4 patches total we were able to improve the performance of filter/action insertion [17] and dump [6] by several orders of magnitude and allow TC to achieve adequate performance when working with hundreds of thousands of filters on a single classifier instance or actions of the same type.

The benefit of using idr for id lookup during filter creation is easy to measure and significantly impacts OvS performance, as seen in Fig. 6. However, the impact of dump performance is indirect since failing to revalidate all flows in some predetermined amount of time (1 second by default [18]) causes OvS to start removing flows from the cache, which causes fluctuations in the total number of offloaded flows and causes handler threads to re-create the flow in the cache.

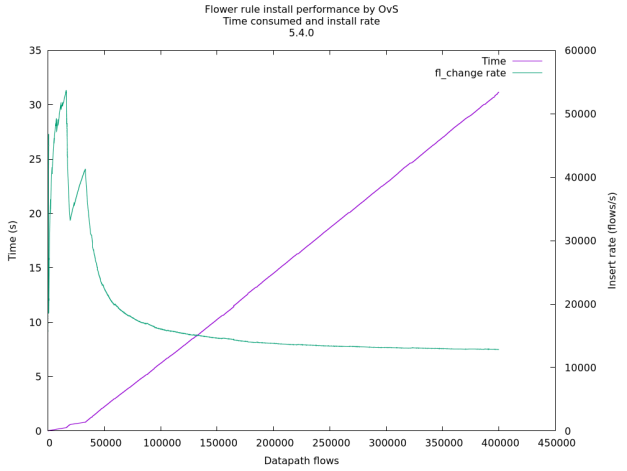

Figure 7: Kernel 5.4 with 16 CPUs available to the host.

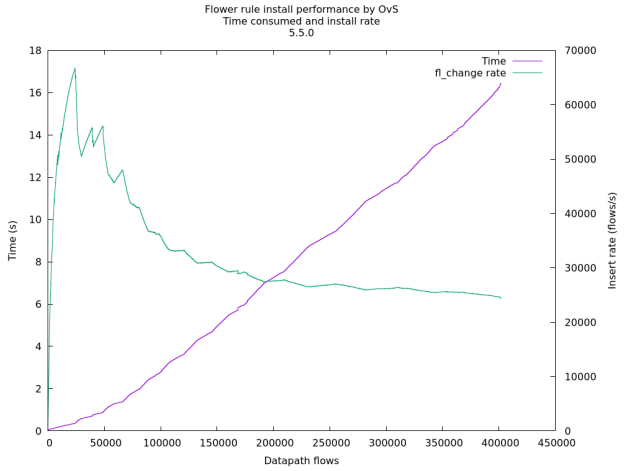

Figure 8: Kernel 5.5 with 16 CPUs available to the host.

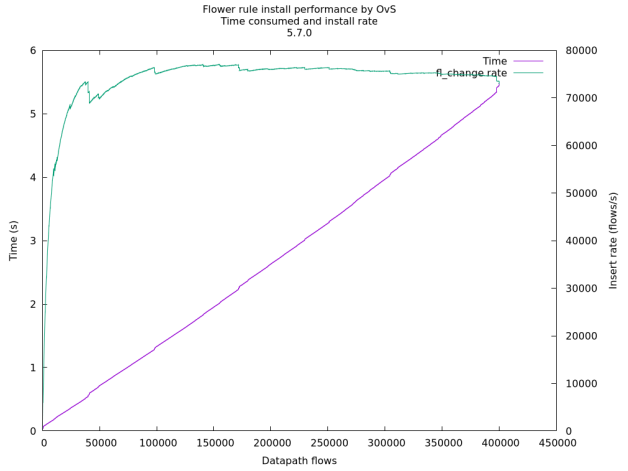

Figure 9: Kernel 5.7 with 16 CPUs available to the host.

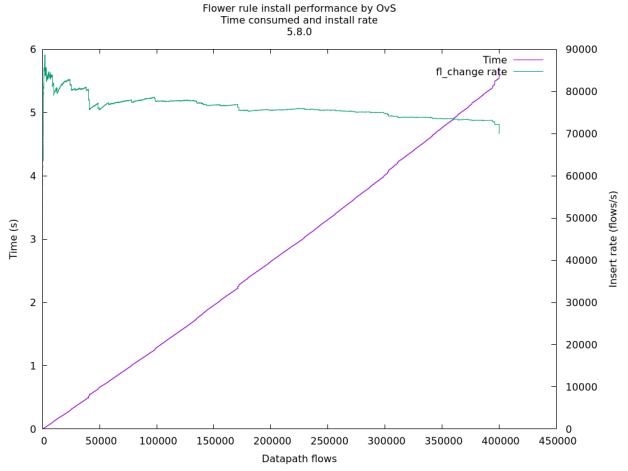

Figure 10: Kernel 5.8 with 16 CPUs available to the host.

Removing rtnl lock dependency

Unlocking TC filter update API proved to be challenging task that required multiple changes in rtnetlink call locking [22], action API [1], all actions implementations [4], ingress Qdisc [3], classifier API [7], flower classifier [9], TC interface to drivers [8] and mlx5 driver TC code [15, 16], which were submitted over the course of two years and resulted in 100+ patches in total. The timeline, details and individual performance impact of the changes had been presented in details on previous TC workshops [2, 11]. The impact it had on OvS can be seen in Fig. 7.

These changes *almost* unlocked execution except when Qdisc or classifier doesn't support unlocked execution, or when reading individual action data during initialization of flow_action intermediate representation for hardware offload. Supporting unlocked execution of flower classifier and ingress Qdisc is enough for offloading OvS TC data path. The last exception (flow_action representation initialization logic) is due to flow_action being introduced concurrently to the upstreaming of multiple patch sets for TC filter update path unlock. Initially, we didn't consider unlocking flow_action initialization code important since it only takes a fraction of percent of filter insertion execution time and didn't introduce rtnl lock contention or impacted performance when benchmarked with iproute2 TC tool in batch mode. However, benchmarks with actual Open vSwitch and traffic showed strange fluctuations in insertion rate. It turned out that even though obtaining rtnl lock for a brief moment doesn't impact concurrent execution of RTM_NEWTFILTER threads, it introduces significant contention in presence of concurrent filter dumps. In hindsight, this interaction should have been obvious just by looking at the OvS threading model interaction with TC Fig. 2, however at the time it caught the author by surprise. The issue was fixed later by completely removing rtnl_lock dependency from flow_action infrastructure [13]. The impact it had on OvS can be seen in Fig. 9.

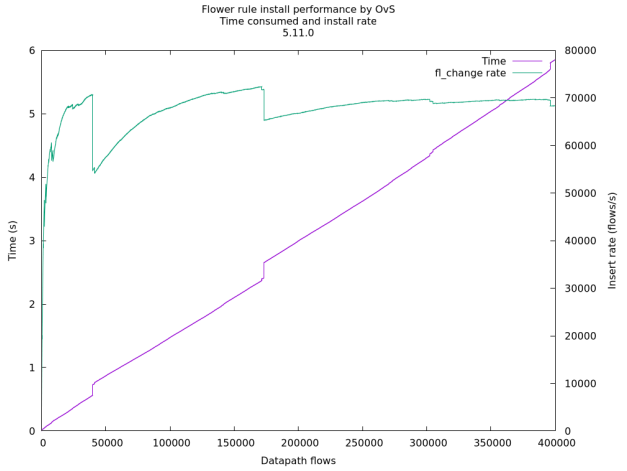

Figure 11: Kernel 5.11 with 16 CPUs available to the host.

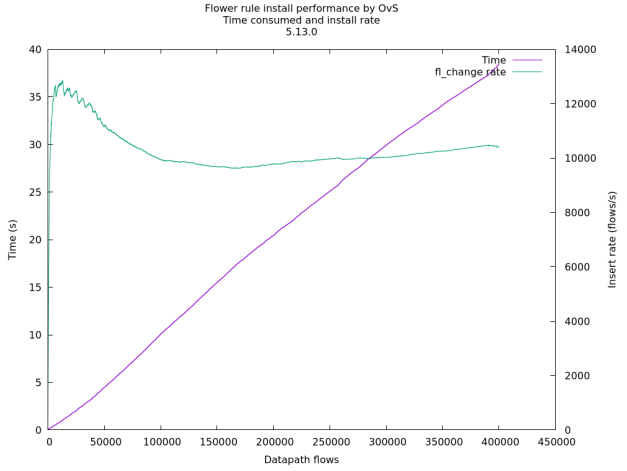

Figure 12: Kernel 5.13 with 4 CPUs available to the host.

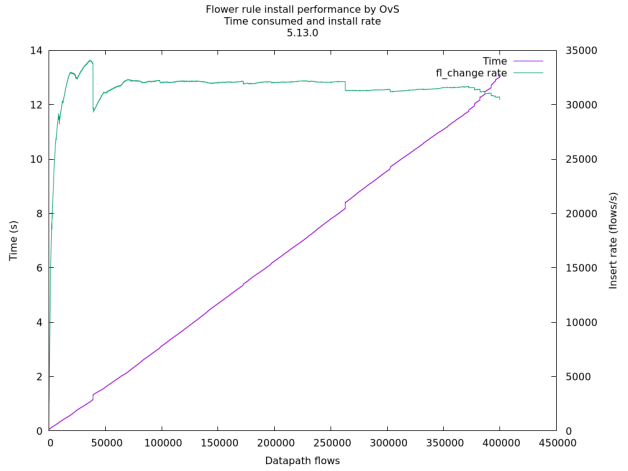

Figure 13: Kernel 5.13 with 8 CPUs available to the host.

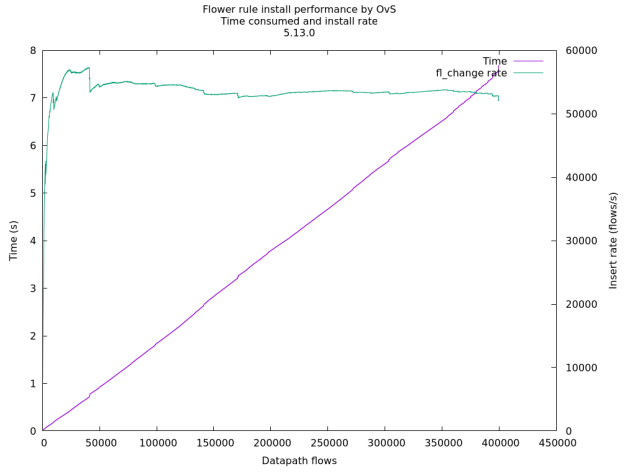

Figure 14: Kernel 5.13 with 12 CPUs available to the host.

Improving TC performance in general

The main issue with TC rule insertion performance was percpu allocator usage. The use of percpu allocator for action statistics was intended as an optimization for the software data path. However, not only offloaded filters don't benefit from this, they actually incur in a performance penalty (both during allocation and when dumping) and in a unjustified increased memory usage. As such, the obvious solution was to introduce a mechanism to skip using percpu allocator for such filters. For this, we introduced action flags netlink attribute and allowed the user to request regular integer fields embedded in action structure for statistics, which significantly improved insertion rate [14] with only 8 patches change in total [5]. The impact it had on OvS can be seen in Fig. 8.

For improving TC dump performance several approaches were considered:

1. Increasing the maximum netlink packet size. That would allow packing more filters to a single netlink packet during dump and improve performance by reducing the number of syscalls needed to send data of all filters on a particular Qdisc or block to user space. However, changing the maximum allowed netlink packet size had been discussed in NetDev mailing list before and there was opposition to the change. Also, the syscall overhead was sizable but not the main CPU consumer when dumping, so the following approaches looked more promising.

2. Allow parallel execution of dump handler by removing rtnl_lock dependency. This approach is similar to improving the rule update rate by parallelizing it as discussed in the previous section. However, it is much more challenging to apply it to dump handlers since we would need a way to somehow partition the filter space (either all filters on Qdisc/block or inside classifier instances). We couldn't come up with a realistic and fast algorithm to implement the partition.

3. Reduce the amount of data included in the dump. TC filter dump serialized and included all the classifier info, ac-

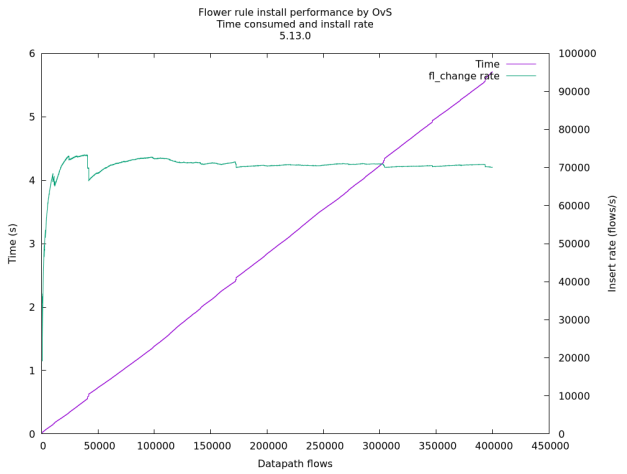

Figure 15: Kernel 5.13 with 16 CPUs available to the host.

tion data and statistics while most of which was ignored by the user space OvS daemon. Significantly decreasing the dump overhead by skipping most of the redundant data looked like the most promising one in terms of performance and straightforward to implement it.

We decided to proceed with the third approach and introduce an alternative dump mode that will have included only the minimum data necessary to identify the flow and its stats. Since OvS already included similar functionality called "terse dump" in other data paths, we decided to reuse all the existing OvS infrastructure and named the kernel flag in a similar fashion TCA_DUMP_FLAGS_TERSE. With only 4 patches in total the filter dump rate was improved by a factor of two [12]. The impact it had on OvS can be seen in Fig. 10.

Issues with ConnectX-5 firmware performance couldn't be resolved and required a different approach to programming NIC steering hardware. The only way to improve performance on existing hardware was to move steering logic into the host mlx5 driver. This became a long-term project called "Software managed steering" that is still ongoing (some big changes after the mode was upstreamed are the usage of buddy-allocator to improve performance and support for new ConnectX-6 NIC) and at the moment includes 150+ feature patches in total. Software managed steering is available since kernel v5.4, while on v5.11 it had the addition of the buddy-allocator. The impact it had on OvS can be seen in Fig. 11.

Currently, with the latest released kernel, v5.13, we can observe how it scales in terms of parallelism in Fig. 12, Fig. 13, Fig. 14 and Fig. 15.

Open issues

Several bottlenecks remain that are either non-trivial to remove or minor enough to ignore for time being:

- Idr data structure used by action API and flower classifier implementation became a bottleneck after applying the optimizations described in this paper. Idr implementation doesn't expose any knobs in order to fine-tune it for our use case. It is also not clear how much performance we can squeeze out of existing kernel idr and radix tree implementation or by implementing some other data structure specifically for TC. More research is required to determine the next steps for this issue.
- Percpu allocator is still used by tunnel_key action indirectly through dst_cache. Since the tunnel_key action is heavily used by the majority of OvS deployments in our experience, it might be worth it to allow the user to disable usage of dst_cache by tunnel_key action. The obvious way to implement it would be by introducing an additional flag similar to the one which disabled percpu action counters.
- Filter update notification generation consumes up to 10% of CPU depending on the use-case. As such, disabling the notification for OvS filters would net significant performance improvement. However, that would prevent any other netlink multicast listeners in the system from being notified about TC state changes for the filters. While such optimization may be successfully used in some specific deployments that require it and where it is known that no other listeners need to be notified about TC filter update, it is not clear that it can be upstreamed. This notification also leads to additional CPU and rtnl_lock usage by NetworkManager (https://bugzilla.redhat.com/show_bug.cgi?id=1753677), as the rate can be excessive and NetworkManager will then dump all filters to update its cache.
- Currently, there is a certain realloc when notifying about the new filter that maybe could be avoided. It happens because tfilter_notify allocates a skb as big as possible and then netlink_unicast calls netlink_trim on it. The improvement out of this one should be small, though. Around 11% of tfilter_notify CPU usage, which can be around 1% of total depending on the test setup.
- Mlx5 software steering, while significantly improving flow update rate compared to firmware-based approach, has its drawbacks. Namely, it consumes the host CPU resources for managing steering hardware and uses a lock for synchronizing its internal state that becomes heavily contended when inserting flows from multiple OvS handler threads. To remedy this we can try to refactor mlx5 software steering code to use finer-grained synchronization mechanisms which may lead to exchanging single-thread performance for better parallelism. Another approach is to fully implement the steering management pipeline in hardware offloading the functionality from both host and embedded NIC CPUs. This would allow us to only leave a thin layer in mlx5 driver that will be responsible for sending messages to a hardware queue and handling the replies and completely bypass complicated hardware resource management. The latter approach is currently being worked on and may be provided by mlx5 driver for future generations of ConnectX SoC.

Conclusions

What we set up to do in 2017 as a couple of quick optimizations to make OvS TC data path performance production-ready turned out to be a long and interesting journey involv-

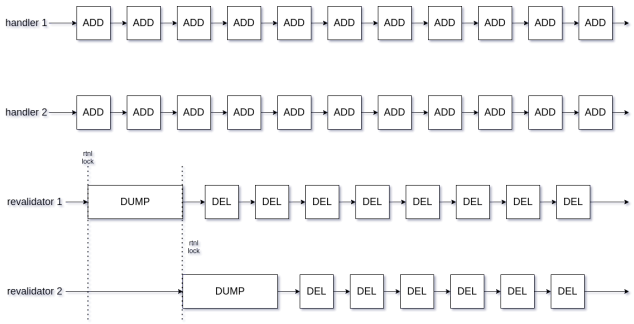

Figure 16: OvS concurrent handlers and revalidators executing with optimizations

ing multiple teams and resulting in hundreds of patches over four years. In the end, while not perfect, TC data path became much more performant and allowed resulting OvS threads execution profile Fig. 16 to closely resemble our idealized version from Fig. 1.

This resulted not only in an improvement of nearly two orders of magnitude in OvS learn rate, going from 800 flows/sec on v4.14 to 70000 on v5.13, with 16 CPUs, but also on a more stable and reliable solution.

We didn't look into OvS specific changes in this paper. It is possible that OvS based changes can help with this matter as well but that was left to another moment.

Acknowledgments

Thanks belong to Jiří Pírko and Jamal Hadi Salim for guiding the author through the depths of TC stack and reviewing tons of scary-looking code; Saeed Mahameed for saving the author from stepping on multiple rakes while unlocking mlx5.

Authors Biographies

Marcelo Ricardo Leitner is a long time Red Hatter, always focused on networking, and amongst other things like stressing the coffee machine he has been leading the inclusion of OvS hardware offload technology on RHEL for the past 4 years.

Vlad Buslov has been dabbling in data paths of all kinds of networking devices for more than ten years. This includes hardware, dpdk, and most recently Linux kernel. He is known to make slow things fast (and sometimes broken).

References

[1] Buslov, V. 2018a. Modify action API for implementing lockless actions. https://lore.kernel.org/netdev/1530800673-12280-1-git-send-email-vladbu@mellanox.com/. Online; accessed 4-July-2021.

[2] Buslov, V. 2018b. Parallelizing TC rules update. https://www.files.netdevconf.info/d/9535fba900604dcd9c93/files/?p=/Parallelizing%20TC%20rules%20update.pdf. Online; accessed 4-July-2021.

[3] Buslov, V. 2018c. Refactor classifier API to work with Qdisc/blocks without rtnl lock. https://lore.kernel.org/netdev/1537806178-10944-1-git-send-email-vladbu@mellanox.com/. Online; accessed 4-July-2021.

[4] Buslov, V. 2018d. Remove rtnl lock dependency from all action implementations. https://lore.kernel.org/netdev/1533923515-5664-1-git-send-email-vladbu@mellanox.com/. Online; accessed 4-July-2021.

[5] Buslov, V. 2019a. Control action percpu counters allocation by netlink flag. https://lore.kernel.org/netdev/20191030140907.18561-1-vladbu@mellanox.com/. Online; accessed 4-July-2021.

[6] Buslov, V. 2019b. net: sched: refactor flower walk to iterate over idr. https://lore.kernel.org/netdev/1531132151-2321-1-git-send-email-vladbu@mellanox.com/. Online; accessed 4-July-2021.

[7] Buslov, V. 2019c. Refactor classifier API to work with chain/classifiers without rtnl lock. https://lore.kernel.org/netdev/20190211085548.7190-1-vladbu@mellanox.com/. Online; accessed 4-July-2021.

[8] Buslov, V. 2019d. Refactor cls hardware offload API to support rtnl-independent drivers. https://lore.kernel.org/netdev/20190826134506.9705-1-vladbu@mellanox.com/. Online; accessed 4-July-2021.

[9] Buslov, V. 2019e. Refactor flower classifier to remove dependency on rtnl lock. https://lore.kernel.org/netdev/20190321131744.19224-1-vladbu@mellanox.com/. Online; accessed 4-July-2021.

[10] Buslov, V. 2019f. tc filter insertion rate degradation. https://lore.kernel.org/netdev/vbfmunui7dm.fsf@mellanox.com/. Online; accessed 4-July-2021.

[11] Buslov, V. 2019g. Unlocking TC rules update API: challanges and lessons learned. https://www.files.netdevconf.info/d/0b87d4f9cca64cd09078/files/?p=/1-NetDev0x13final.pdf. Online; accessed 4-July-2021.

[12] Buslov, V. 2020a. Implement classifier-action terse dump mode. https://lore.kernel.org/netdev/20200515114014.3135-1-vladbu@mellanox.com/. Online; accessed 4-July-2021.

[13] Buslov, V. 2020b. Remove rtnl lock dependency from flow_action infra. https://lore.kernel.org/netdev/20200217101212.5979-1-vladbu@mellanox.com/. Online; accessed 4-July-2021.

[14] Buslov, V. 2020c. TC performance update. https://legacy.netdevconf.info/0x14/pub/slides/13/TCperf-0x14.pdf. Online; accessed 4-July-2021.

[15] Mahameed, S. 2019a. Mellanox, mlx5 tc flow handling for concurrent execution (Part 1). https://lore.kernel.org/netdev/20190729234934.23595-1-saeedm@mellanox.com/. Online; accessed 4-July-2021.

[16] Mahameed, S. 2019b. Mellanox, mlx5 tc flow handling for concurrent execution (Part 2). https://lore.kernel.org/netdev/20190809220359.11516-1-saeedm@mellanox.com/. Online; accessed 4-July-2021.

[17] Mi, C. 2017. net/sched: Improve getting objects by indexes. https://lore.kernel.org/netdev/1502871244-19870-1-git-send-email-chrism@mellanox.com/. Online; accessed 4-July-2021.

[18] Pfaff, B.; Pettit, J.; Koponen, T.; Jackson, E.; Zhou, A.; arno Rajahalme; Gross, J.; Wang, A.; Stringer, J.; Shelar, P.; Amidon, K.; and Casado, M. 2015. The design and implementation of open vswitch. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*.

[19] Pírko, J. 2015. Implementing open vswitch datapath using tc. In *Proceedings of NetDev 0.1*.

[20] Salim, J. H. 2015. Linux traffic control classifier-action subsystem architecture. In *Proceedings of NetDev 0.1*.

[21] Shattah, G., and Efraim, R. 2017. Improving tc filters insertion rate. In *Proceedings of NetDev 2.2*.

[22] Westphal, F. 2017. rtnetlink: allow selected handlers to run without rtnl. https://lore.kernel.org/netdev/20170809184153.16700-1-fw@strlen.de/. Online; accessed 4-July-2021.

[23] Zhou, D. 2019. introduce percpu block scan_hint. https://lore.kernel.org/lkml/20190228021839.55779-1-dennis@kernel.org/. Online; accessed 4-July-2021.