Implementing Open vSwitch datapath using TC

Jiří Pírko

Red Hat
Prague, Czech Republic
jiri@resnulli.us

Abstract

This is a proposal to illustrate how to implement the OVS DP (Open vSwitch kernel datapath) using the Linux TC (traffic control) subsystem. The TC subsystem existed long before OVS DP and offers more flexibility. As an example it allows creating multiple types of classification which can be added as plugins. It offers capability of working on the ingress or egress path of any Linux netdev which means the datapath does not require the central focus to be a switch.

This talk will go into covering the missing classificators and actions that are implemented in OVS DP and needed by TC to achieve this goal. The talk will describe recent updates being done and also the planned ones to add code-reusability to some of the OVS DP actions and how they are used by tc to achieve the stated goal.

Keywords

TC, classifier, action, OVS, datapath

Introduction

The goal of this paper is to provide an alternative approach to what is currently implemented as OVS DP (Open vSwitch[1] kernel datapath).

The main reason for this is the increasing amount of features that have been implemented in multiple kernel parts. OVS DP code contains a lot of code duplication. Would be nice to reduce the duplication in the future which would make the maintenance easier, bug amount lower and user happier.

Another consequence of moving to TC (traffic control) would be easier offload to hardware possibilities. It is a very complex problem to offload flow-based forwarding. Better to do it once and for all.

This paper will go through the items that need to be added to TC in order to provide the same features the current OVS DP offers.

Open vSwitch kernel datapath overview

The OVS DP is a match-action forwarding datapath. The information about the match and actions to be taken on the packet which matches are in a form of “flows”. These flows are inserted, modified or removed by userspace.

The flow format is inherited from OpenFlow (specification of Open Networking Foundation[2]), as OVS DP is heavily influenced by that. Each flow consists of three things:

- Flow key – according to OpenFlow, this key represents a set of fields in a packet on which the datapath should match.
- Flow mask – this is a bitmask for flow key. User can use this to do wildcard matches.
- Set of actions – user can specify a set of various actions to be executed in case the packet matches the key with mask. This includes outputting the packet to a certain port or to userspace, pushing and popping VLAN headers, etc.

OVS DP has a notion of so called vports (virtual ports). There are essentially two kinds. Ones are backed-up by real netdevice, the others are not and serve mainly as a place holder for tunnels.

Multiple vports are put into groups called bridges. Each bridge is represented by an “internal” vport which is backed up by a special kind of OVS DP netdevice.

Another very important part of the datapath implementation is the possibility to create various tunnels, including GRE, Geneve and VXLAN tunnels. These tunnels are not represented by a separate netdevice, they are not visible from outside Open vSwitch infrastructure.

Classifier-Action subsystem of TC

TC already contains Classifier-Action subsytem[3]. The referenced paper goes into details so the author recommends the reader to study that.

This subsystem provides possibility to use various kinds of classifiers which do the matching of packets based on various things. Here is an example of couple of classifiers:

- u32 – allows to match packets based on key, mask and offset values.
- bpf – uses a BPF filter to match packets.

According to the match a chain of actions can be executed. Here is an example of couple of action:

- skbedit – allows to exit predefined skb fields.
- nat – allows to do stateless NAT.
- mirred – allows to mirror or forward packet to particular netdevice.
- vlan – allows to push an pop VLAN header.

Using Classifier-Action to implement OVS DP

According to the similarity of the packet matching and action processing in both TC Classifier-Action (will refer to this as TC CA later in the text) subsystem and OVS DP, one could ask if one can be replaced by another.

TC CA is around for a very long time and it is very flexible and easily extendable. Many of the existing TC CA features which are needed to implement OVS DP are in place. Some of them are not and need to be added.

For implementing the OVS DP functionality, the TC classifier should be attached to ingress qdisc. The Listing 1 shows very trivial forwarding example. It results into a blind forward of every packet incoming through one netdev to another one and vice versa.

```
tc qdisc add dev eth0 ingress
tc filter add dev eth0 parent ffff: protocol all u32 match u32 0 0 \
        action mirred egress redirect dev eth1
tc qdisc add dev eth1 ingress
tc filter add dev eth1 parent ffff: protocol all u32 match u32 0 0 \
        action mirred egress redirect dev eth0
```

Listing 1: Simple forwarding example

The classifier

The mentioned u32 classifier is rich enough to cover the match features included in OVS DP. But the generic approach u32 classifier has some downsides. For one, there has to be a wrapper which would map each individual key item from OVS DP to u32 key, mask, offset tuple. Also, the generic approach has performance impacts.

Since there is easy to add a new classifier, the author started to work on a new classifier called cls_openflow. It follows the OpenFlow specification in implementation of the essential key items. OVS DP implements couple of optional items on top of that. It is planned to support them in cls_openflow as well.

Currently the cls_classifier supports to match on following fields:

- Input device
- Destination Ethernet address
- Source Ethernet address
- Ethernet Type
- IP protocol
- IPv4 and IPv6 source address
- IPv4 and IPv6 destination address
- TCP and UDP source port
- TCP and UDP destination port

The cls_openflow classifier is not yet included into mainline kernel, the plan of the author is to push it very soon.

```
tc qdisc add dev eth0 ingress
tc filter add dev eth0 parent ffff: protocol all openflow \
        src_ip 192.168.0.1 dst_ip 192.168.10.0/24 \
        action mirred egress redirect dev eth1
```

Listing 2: Simple use of cls_openflow classifier example

In Listing 2 there is a simple example of use of cls_openflow classifier. The packets incoming via eth0 with source IP address 192.168.0.1 and destination IP address 192.168.10.0/24 will be matched. The mirred action will be executed then to forward the packet using eth1 netdevice.

Output action

The output action in OVS DP results in a transmission of a packet using specified port.

In TC CA, there is a mirred action which is very suitable to cover this action functionality. Reader can see the usage of mirred in Listing 1 and Listing 2.

Upstream output action

In OVS DP in case an incoming packet does not match any inserted flow (this is also called “flow-miss”), it is ejected into userspace. The userspace daemon inspects the packet, inserts the appropriate flows into kernel datapath and re-inserts the packet into the datapath. This is done by passing packet using generic Netlink interface.

In TC CA, this can be implemented using tap interface. There is only need to create one tap device for one bridge. When the flow-miss happens, the packet will be forwarded to the tap device using mirred action. The userspace daemon will receive this packet via character device associated with the tap device.

VLAN header pop and push actions

For the purpose of manipulation of VLAN headers using TC CA the author introduced new action called “vlan”. It allows to pop existing VLAN header. It also allows to push a new VLAN header with specified tag and protocol type.

Listing 3 shows an example where every packet incoming via eth0 will be stripped of the original VLAN header, new VLAN header with ID 100 will be inserted and the packet will be forwarded using eth1 netdevice.

```
tc qdisc add dev eth0 ingress
tc filter add dev eth0 parent ffff: protocol all u32 match u32 0 0 \
        action vlan pop \
        action vlan push id 100 \
        action mirred egress redirect dev eth1
```

Listing 3: VLAN header pop and push example

MPLS header pop and push actions

Similar to VLAN, MPLS support in OVS DP is also implemented as a fixed sized header added beyond the Ethernet header.

At the time this paper was written, there was no support for MPLS header in TC CA. It should be fairly easy to add this support though.

Tunneling

This subsection focuses on support for VXLAN, Geneve and GRE tunnels. It's very likely that more tunnel types are going to appear in the future.

In OVS DP, these tunnels are represented using virtual ports. There are no backing netdevice for them. They are all backed just by a socket. The socket is created when a vport is created, it registers a receive function and there is a transmit function as well. Currently, OVS DP uses following transmit functions:

- vxlan_xmit_skb
- geneve_xmit_skb
- iptunnel_xmit

The first possible solution for TC CA is to create a netdevice for each individual tunnel. After that, the tunnel netdevice would be treated as any other netdevice. The mirred action can be used to forward packets into tunnels.

There are two issues to this approach. First, not all of the tunneling methods can be used as a separate netdevice. For example Geneve standalone netdevice implementation is not present. Second, there are some scalability issues connected with that. There are use cases when user dynamically creates and destroys thousands of tunnels and adding and removing netdevices for every of them is not feasible. That is the reason to introduce a way to handle tunnels in TC CA without having a netdevice for each tunnel.

One alternative would be to implement tunneling with actions. Upon the action creation the tunnel socket would be created. Upon the action execution the appropriate tunnel transmit function would be executed.

Problematic part of this approach is the receive side. Since there is not a netdevice to attach classifier to, TC CA subsystem cannot process the incoming packets. It would be possible to introduce a new classifier for tunnel matching. Then, the user would attach the classifier to the device on which the tunneled traffic is coming from. That would lead to unnecessary code duplication.

The second approach would be to extend the existing interface to provide possibility to create tunnel sockets from userspace. These sockets would be named. TC CA interface would be extended to allow to attach ingress qdisc not only to a netdevice, but also to these named tunnel sockets. That resolves the receive path.

Having these special sockets present, the specific tunnel action can be avoided. Instead of it, mirred action could be extended to forward packets not only to a netdevice, but also to named tunnel sockets.

```
ip link add vxlansock0 type vxlan \
        id 42 group 239.1.1.1 dev eth1 sock
tc qdisc add dev eth0 ingress
tc filter add dev eth0 parent ffff: protocol all \
        u32 match u32 0 0 \
        action mirred egress redirect sock vxlansk0
tc qdisc add sock vxlansock0 ingress
tc filter add sock vxlansock0 parent ffff: protocol all \
        u32 match u32 0 0 \
        action mirred egress redirect dev eth0
```

Listing 4: Example of use of named VXLAN socket

Listing 4 shows hypothetical usage of named socket. First, the vxlan tunnel is created using ip tool. Option “sock” tells kernel not to create netdevice but only a “named socket”. Then the transmit side is handled by redirecting data incoming from eth0 to the socket. Then the ingress qdisc is attached to the socket and received packets are redirected to eth0.

Note that this is just an idea, there is no actual implementation done.

Other approaches

TC CA is not the only possibility to implement an alternative to OVS DP. It would be also possible to use nftables[4] for the same thing. Actually a lot of functionality of nftables and TC CA subsystem also duplicates.

Another approach would be to use eBPF[5] to do match-action based forwarding.

Conclusion

When all needed parts are implemented in TC CA, userspace daemons would be able move from the OVS DP to TC CA. That of course allows it to use not only the things introduced for OVS DP, but also a lot from the existing stuff, like various classifiers and actions.

For certain, there will be necessary to face and resolve many issues. For example locking overhead in current TC CA will need to be removed.

Acknowledgements

Thanks belong to Jamal Hadi Salim for opening author's eyes to see that OVS DP is unnecessary and also to Jiří Benc who has been consulting the solution with the author.

References

1. Open vSwitch website, http://openvswitch.org/
2. Open Networking Foundation website, https://www.opennetworking.org/
3. Jamal Hadi Salim, "TC Classifier Action Subsystem Architecture", Proceedings of Netdev 0.1, Feb 2015
4. nftables project website, http://netfilter.org/projects/nftables/
5. BPF kernel documentation https://www.kernel.org/doc/Documentation/networking/filter.txt

Author Biography

Jiří Pírko is a Linux kernel hacker who has been digging in networking subsystem for some while. He is an author of Team device, a bonding replacement. He is also a co-author of Rocker switch and its support in Linux kernel including the switchdev infrastructure. His life purpose is to keep things open, nice, clean and easy.