Traffic Control connection tracking hardware offload

Oz Shlomo, Paul Blakey

Mellanox, Ra'anana Israel
ozsh@mellanox.com, paulb@mellanox.com

Abstract

Recent industry advancements toward the use of Traffic Control (TC) as well as Open vSwitch (OVS) has created new requirements, one of which is supporting connection tracking (CT). Connection tracking datapath actions and matches require stateful packet processing that is challenging to fully offload. However, there is still significant value in processing connections in established state, where the bulk of the connection data is passed, in hardware while handling the connections' setup and teardown, which cause connections to enter and exit the established state, on the software slow path. In addition, packet steering based on the CT state requires packet reclassification, which dictates a multi-table design. Hardware offload of multi-table rules exposes the system to misses in hardware because the hardware offload process is not atomic. In this paper, we will present an infrastructure that was recently introduced to the Linux kernel to support the hardware offload of the connection tracking action.

Keywords

Linux, Traffic Control (TC), Connection tracking (CT), conntrack, nft, netfilter (nf), flowtable (ft), hardware offload, Openvswitch (OVS)

Introduction

Connection tracking is a tool for stateful packet processing [1]. Connections are managed in a flow table enabling the association of a packet with the following states:

- New – first packet of a connection.
- Established – two-way communication was established.
- Related – packets of a connection that are part of existing related connection (e.g. FTP data connection)
- Invalid – packet that cannot be associated with an existing connection. For example, TCP packets on established connections are not within the expected sliding window. Note that this functionality may be controlled using the tcp_liberal flag.

Connections may also be associated with a user-defined 32-bit mark or 128-bit label values. These values may be used by the user for filtering application logic. Source/destination Network Address Translation (SNAT/DNAT) may also be applied.

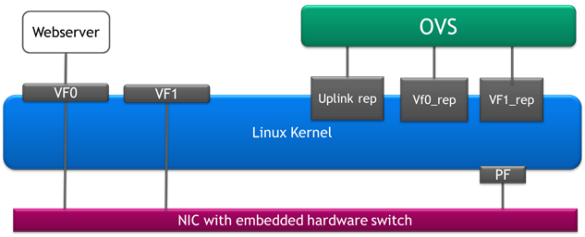

Figure 1. Switchdev setup

Several user-space applications such as iptables, nft and Open vSwitch (OVS) integrate with the connection tracking module to provide network security functionality. For example, OpenStack realizes the security groups feature using connection tracking as the tool to provide stateful ingress/egress access control.

Connection tracking action was recently added to Linux TC [2]. The ct action uses the nf conntrack module to initialize the connection tracking state on the skb. In addition, the tc flower classifier was enhanced with the ability to match on the ct state, mark and label fields, thus providing the ability for stateful packet steering using tc [3].

Let's consider an example of applying a network security policy using tc. Figure 1 illustrates a switchdev setup where a virtual machine (VM) is running a webserver on top of VF0. The network administrator may apply a security policy allowing ingress http traffic to the vm using the following set of rules on the uplink port:

```
$ tc filter add dev uplink_rep ingress prio 1 chain 0
proto ip flower ct_state -trk ip_proto tcp dst_port 80
action ct pipe action goto chain 1

$ tc filter add dev uplink_rep ingress chain 1 prio 1
proto ip flower ct_state +new+trk action ct commit
pipe action mirred egress redirect dev vf0_rep

$ tc filter add dev uplink_rep ingress chain 1 prio 1
proto ip flower ct_state -new+est+trk action mirred
egress redirect dev vf0_rep
```

Figure 2. Security group ingress filters

The first rule classifies http traffic by matching on tcp destination port 80. Matching packets are processed by the CT action which initializes the connection's state on the skb. The packets with the updated ct state are then reclassified on chain 1. Chain 1 has two flows, each matching on different ct state. The first packet of a new connection, matched by the second rule, is committed to the ct database and forwarded to the vm representor port. Packets on established connections, matched by the third rule, are simply forwarded to the vm representor ports. The VM egress packets, received on the vf0 representor port, follow the following rules:

```
$ tc filter add dev vf0_rep ingress prio 1 chain 0 proto ip flower ct_state -trk ip_proto tcp dst_port 80 action ct pipe action goto chain 2

$ tc filter add dev vf0_rep ingress chain 1 prio 1 proto ip flower ct_state +est+trk action mirred egress redirect dev uplink_rep

$ tc filter add dev vf0_rep ingress chain 1 prio 1 proto ip flower ct_state +new+trk action drop
```

Figure 3. Security group egress filters

The first filter also performs a ct lookup. However, packets that are not part of an established connections are dropped, as defined in the third rule. This ensures that reply packets were initiated by a client request.

TC Offloads

TC filters are offloaded to hardware at the time of their instantiation. When a tc filter is created, registered driver callback methods are invoked with the filter's list of match criteria and its corresponding actions. With this input, the device driver can offload the given rule using the hardware's data model. Network device drivers may register to receive callbacks when filters are added or deleted, either directly on their network devices or indirectly on upper layer network device (e.g. tunnel network device). The tc *in_hw* flag is set when the filter was successfully offloaded to hardware.

Offloading CT Actions

The CT action cannot be fully offloaded because current hardware does not have the stateful engines allowing it to replicate nf conntrack software logic. However, no stateful processing is required once connections enter the established state (i.e. traffic was processed by both directions), other than tcp window validation that can be controlled using the tcp_liberal flag. As such, the system is designed in a way where established connections are offloaded to hardware while connection setup, teardown and aging is handled by software.

Offloading only the established connections requires the platform to notify the relevant device drivers when connections enter or exit the established state. Aged connections should also be removed from hardware.

Several integration points can be considered:

- Netfilter conntrack – the conntrack module manages the connection state changes and has an aging engine.
- TC connection tracking action – the action uses nf conntrack module. As such, the ct action can deduce state changes from the conntrack return value.
- Netfilter flowtable offloads – nft mechanism that is used for bypassing the classic forwarding path for established connections [4, 5]

The nf flow table API was generalized to provide the ability to create, delete and lookup flow table entries while notifying registered network drivers of any add/del event [6]. The nf flowtable offload infrastructure already has an aging mechanism which integrates with netfilter conntrack aging via the IPS_OFFLOAD_BIT flag. Connection aging events are also communicated to the registered network drivers. Note that the same nf flow table API may also be used for the hardware offload of nft flow table offloads [7].

Offloading established connections in TC

The connection tracking tc action integrated with the new nf flow table API by maintaining an nf flowtable instance per zone [8]. Flow table entries are added when TCP/UDP connections enter or exit the established state accordingly. Connection aging is managed by nf flowtable and is transparent to action ct.

The nf flowtable may also accelerate the software processing of action ct as the flow table lookup is faster than nf conntrack processing. As such, packets that are processed in software may set the ct info on the skb by executing a nf flow table lookup [9]. Lookup misses, for connections that are not in the established state yet, or not in the flow flow table, will be processed by the nf conntrack module as usual.

The nf flow table instances also serve as the integration platform for the hardware offload of established connections. While offloading the CT action to drivers via flow offload API, the flow table instance is provided, allowing the network device driver to register a flow add/del/stats callback on it. When a connection enters the established state, a 5-tuple entry is instantiated, and the driver is called back with a *flow_offload* object containing the following input:

- 5-tuple match

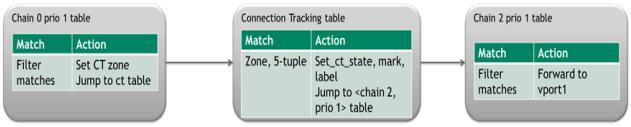

Figure 4. Connection tracking hardware model

- ct_metadata action - containing a reference to the ctinfo object along with the connection's zone, mark and label fields
- Array of packet mangle actions if snat/dnat operations are required.

Once a driver is notified of a new flow table entry it can extract from the ct metadata action the ct state, zone, mark and label field and perform snat/dnat if needed. Hardware can then move on to next requested tc action - usually re-classification on the next chain. Figure 4 illustrates a multi-table hardware offload model as implemented in [11].
The del callback is invoked when a connection is deleted from the relevant flow table, either from a tcp teardown process, connection aging timeout, or flow table purge/deletion. Aging periods are managed by the nf flow table autonomously from nf conntrack aging and they are currently hardcoded to 30 seconds. Future work may control the flow table aging period.

Miss handling

Connection tracking action in the tc and ovs datapaths initializes the ct state and called explicitly when required. The default uninitalized ct state is untracked (-trk). Once a packet goes through the conntrack module, it is now marked as tracked (+trk), and the connection's state is updated (+new, -new, +est, -est, etc). This paradigm requires a multi-table architecture where the matching phase needs to be repeated after the ct action is processed. Multi-table designs can be implemented in tc using the *goto chain* action or in openvswitch using the *recirc* action. A simple ct multi-table classification in tc and openvswitch datapath are shown in figures 5 and 6 accordingly.
Openvswitch dpctl rules are translated to tc flower filters whenever the openvswitch hardware offload setting is enabled.
Openvswitch dpctl feature set supersets tc's feature set. As such, not all dpctl rules can be translated to tc filters. This introduces a scenario where a tc goto chain action may miss because the next chain's filter was not translated to tc. In such use cases, the packet will continue to the openvswitch rx handler, where the processing will complete. To support such scenarios, tc marks the last chain that was processed by it on an skb extension field. Openvswitch then starts its processing with this corresponding recird_id [10].

```
recirc_id(0),in_port(1),ct_state(-trk),…, actions:ct,recirc(3)
recirc_id(3),in_port(1),ct_state(+new+trk),…, actions:ct(commit),3
recirc_id(3),in_port(1),ct_state(-new+est+trk),…, actions:4
```

Figure 5. OVS dpctl dump extract

```
tc filter … chain 0 … flower ct_state -trk …. \
        action ct pipe action goto chain 3
tc filter … chain 3 … flower ct_state +new+trk …. \
        action ct commit pipe action mirred …
tc filter … chain 3 … flower ct_state -new+est+trk …. \
    action mirred egress redirect dev port4
```

Figure 6. TC filter show output extract

The packet recirculation software model also applies to the hardware model. TC <chain, prio> tuples are uniquely mapped to a table in hardware. TC *goto chain* action is translated to a *jump table_id* action in hardware. Not all the TC filters may be offloaded to hardware either because the offload process is pending, or the hardware may lack the capabilities to offload the filter's matches and actions. As such, the packet processing which started in hardware may miss and software is expected to continue the processing from the tc chain id where the hardware missed. It is incorrect sto re-start the processing from chain 0 as the matched filters already counted the packets and may have manipulated them. The TC skb extension is reused for drivers to communicate to tc what was the last chain which was processed in hardware, and then tc continues processing from that chain.
Hardware misses may also restore the tunnel information, which may have been removed on the hardware *decap* action, and the connection tracking state, as the hardware may have already processed the ct action. High level hardware offload architecture may be reviewed at [11].

Future work

Connection tracking offloads introduce new scalability requirements both in terms of table size (order of 1M connections) and insertion rate (100s K connections per second). New platform components may be required to support and manage offload related bottlenecks.
Internet traffic flow size and duration have heavy tail distribution. Most of the flows are very short while most of the traffic bandwidth is carried by a small number of connections. Restricting the hardware offload to elephant flows can drastically reduce the scale requirements while still offloading a very large percentage of the traffic. In addition, on heavily loaded systems, short lived connections may be closed by the time they are offloaded. Currently the system is under the working assumption that everything should be loaded. Under this assumption, the software issues an offload work item for every packet with an established state. As such, if the hardware fails to offload a connection then, in effect the software would re-offload that connection for every packet it will process.

Perhaps a mechanism should be considered to suppress the re-offload events under certain conditions.

References

[1] Pablo Neira Ayuso, “Netfilter’s connection tracking system”, :login; the USENIX magazine. JUNE 2006

[2] Blakey P. Leitner M. Kuperman Y. commit b57dc7c1 “net/sched: Introduce action ct” https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net.git/commit/?id=b57dc7c13ea90e09ae15f821d2583fa0231b4935

[3] Guy Shattah, Rony Efraim Extend TC to support Connection Tracking, Proceedings of Netdev 2.2, Feb 2017

[4] Linux kernel documentation on Netfilter flowtable: https://www.kernel.org/doc/Documentation/networking/nf_flowtable.txt

[5] Pablo Neira Ayuso - Flow offload infrastructure, https://lwn.net/Articles/738214/

[6] Pablo Neira Ayuso commit c29f74e0 “netfilter: nf_flow_table: hardware offload support” https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net.git/commit/?id=c29f74e0df7a02b8303bcdce93a7c0132d62577a

[7] Paul Blakey commit 8417998 “net/mlx5: TC: Offload flow table rules” https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net.git/commit/?id=84179981317fb4fb3e9df5acd42ea33cf6037793

[8] Paul Blakey commit c34b961a “net/sched: act_ct: Create nf flow table per zone” https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net.git/commit/?id=c34b961a249211bdb08d03bdecfb31ff22eb002f

[9] Paul Blakey commit 46475bb2 “net/sched: act_ct: Software offload of established flows” https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net.git/commit/?id=46475bb20f4ba019abf22b0db10bf55a4158852e

[10] Paul Blakey, Vlad Buslov commit 95a7233c “net: openvswitch: Set OvS recirc_id from tc chain index” https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net.git/commit/?id=95a7233c452a58a4c2310c456c73997853b2ec46

[11] Paul Blakey - Introduce connection tracking offload, https://lwn.net/Articles/814061/