TC Flower Offload

Simon Horman
Netronome
simon.horman@netronome.com

Abstract

The TC Flower Classifier allows control of packets based on flows determined by matching of well-known packet fields and metadata. This is inspired by similar flow classification described by OpenFlow and implemented by Open vSwitch. Offload of the TC Flower classifier and related modules provides a powerful mechanism to both increase throughput and reduce CPU utilisation for users of such flow-based systems. This presentation will give an overview of the evolution of offload of the TC Flower classifier: where it came from, the current status and possible future directions.

Introduction

Packet processing may be viewed as a sequence operations a packet passes through from ingress to its ultimate destination; typically egress or being dropped. Historically many datapath implementation have been fixed-function, arranged into tables where packets are successively matched, acted on using actions and optionally passed to a subsequent table.

In one sense such an architecture is a sequence of one or more match/action rules, where a packet classifier seeds a flow key describing the packet and determines if any of the rules present in the table match the packet. If so the actions associated with the rule are executed. If not fall-through occurs, for example the passing the packet to the next predefined processing stage.

The TC flower classifier, part of the TC subsystem in the Linux kernel, provides a mechanism to describe matches on packets using a flow key it defines. Currently the flow key may include both fields extracted from packet fields and optionally tunnel metadata. TC actions may be used to perform drop, modify, output and a variety of other operations on packets.

This mechanism is similar to that described by OpenFlow and implemented in Open vSwitch. It is the authors understanding that TC Flower was originally designed to some extent with them in mind.

Flower Implementation Overview

Flower makes use of the Linux flow dissector to extract packet data into a flow key. The keys of interest are passed as parameters to the flow dissector and only these keys are populated by it in the resulting flow key.

The populated flow key is masked and matched against the rules of the classifier. If a match is found then actions of the matching rule, if any, are executed.

The implementation currently only allows only one mask per priority-level and matches against different masks must be assigned to classifiers at different priority levels. If matches are not disjoint then some care must be taken in ordering the priorities of classifiers accordingly.

As of v4.14-rc4 the supported flow key matches are as follows.

Metadata Fields:

Field	Maskable
Input Device	No

Unlike other matches, the input device is not currently populated by the flow dissector. Rather it is populated directly from a packet's skb metadata.

Packet Data Fields:

Field	Maskable
Ethernet Source Address	Yes
Ethernet Destination Address	Yes
Ethernet Type	No
IP Protocol	No
IPv4 or IPv6 Source Address	Yes
IPv4 or IPv6 Destination Address	Yes
TCP, UDP or SCP Source Port	Yes
TCP, UDP or SCP Destination Port	Yes
VLAN ID	No
VLAN Priority	No
ICMPv4 or ICMPv6 Type	Yes
ICMPv4 or ICMPv6 Code	Yes
ARP SIP	Yes
ARP TIP	Yes
ARP Op	Yes
ARP SHA	Yes
ARP THA	Yes
MPLS TTL	No
MPLS BoS	No
MPLS TC	No
MPLS label	No
TCP Flags	Yes
IPv4 or IPv6 TOS	Yes
IPv4 or IPv6 TTL	Yes

Tunnel Metadata Fields:

Field	Maskable
Encap. Key ID	No
Encap. IPv4 or IPv6 Source Address	Yes
Encap. IPv4 or IPv6 Destination Address	Yes
Encap. UDP Source Port	Yes
Encap. UDP Destination Port	Yes
Encap. Flags	Yes

Encapsulation flags is used to allow a match to differentiate between fragments and non-fragments.

Like other TC classifiers flower may be used in conjunction with a range of actions including csum, gact, mirred, pedit, skbmod, tunnel key and vlan.

An example usage of the TC flower classifier is as follows, it drops IPv4/SCTP packets received on eth0 whose destination port is 80.

```
# tc qdisc add dev eth0 ingress
# tc filter add dev eth0 protocol ip \
    parent ffff: \
      flower ip_proto sctp \
        dst_port 80 \
          action drop
```

Hardware Offload

Motivation

In the case of the flower (and BPF) offload implemented in the Netronome NFP driver the aim is to give both improved throughput across a variety of work-loads.

History

The git kernel history indicates that the origins of the current support for TC hardware offload lie in work by John Fastabend to add support for offloading to HW based QoS schedulers[1]. This work introduced ndo_setup_tc and was accepted for inclusion in v2.6.39 in January 2011.

In February 2016 John Fastabend added further HW offload support extending the use of ndo_setup_tc to allow offload of TC classifiers. This included an offload of the u32 classifier[2]. This work was included in v4.6.

In v4.14-rc4 ndo_setup_tc is used to offload the BPF, flower, u32 and matchall classifiers as well as mqprio.

Mechanism

When a TC flower rule is added flower determines the offload device, if any, of the flow. An offload device is one that has the NETIF_F_HW_TC feature enabled and implements ndo_setup_tc. First the device to which the rule is attached is checked and if it is an offload device it is used. If not the actions of the rule is scanned for a mirred action (that outputs to a device). The device of the first such action is then checked to see if it is an offload device. If so it is used, if not the flow is not offloaded.

The above scheme works for cases where rules are added directly to an offload device, for example dropping packets received on a device with offload capabilities. And adding rules to software devices which output to an offload device, for example tunnel decapsulation followed by output to a device with offload capabilities. It does not, however, handle adding rules to software devices that also output to software devices.

If an offload device is found rule is passed to its ndo_setup_tc implementation. This implementation may reject a rule for any reason including that any part of match, actions or combination thereof is not supported; and lack of space on the device for a new rule. This allows drivers and moreover offload devices themselves to ultimately determine which rules are not offloaded.

Policy

Hardware offload may be enabled and disabled on a per-netdev basis using the NETIF_F_HW_TC feature which may be toggled using ethtool:

```
# ethtool -K eth0 hw-tc-offload on
# ethtool -K eth0 hw-tc-offload off
```

Hardware offload may also be controlled on a per-rule basis using the TCA_CLS_FLAGS_SKIP_HW and TCA_CLS_FLAGS_SKIP_SW flags. These flags are mutually exclusive.

- **skip-hw** denotes that the rule will be added to software but not hardware. An error is reported if adding the flow cannot be added to software.
- **skip-sw** denotes that the rule will be added to hardware but not software. An error is reported if adding the flow cannot be added to hardware.
- **The default** behaviour is to try to add the rule to both hardware and software. No error is returned if the flow cannot be added to harware but error is reported if the flow cannot be added to software.

Two flags, TCA_CLS_FLAGS_IN_HW and TCA_CLS_FLAGS_NOT_IN_HW are provided to allow the kernel to report the presence of a rule in hardware. The tc command-line tool makes use of the TCA_CLS described above to allow the user to control and inspect the placement of rules in hardware and software. The example below uses the skip_sw parameter to tc to add a rule to hardware but not software.

```
# tc qdisc add dev eth0 ingress
# tc filter add dev eth0 protocol ip \
    parent ffff: \
      flower skip_sw \
        ip_proto sctp dst_port 80 \
          action drop
```

This is reflected in the flow reported. in_hw indicates unambiguously that the flow is present in hardware.

```
# tc filter show dev eth3 ingress
filter parent ffff: protocol ip
  pref 49152 flower chain 0
filter parent ffff: protocol ip
 pref 49152 flower chain 0
 handle 0x1
  eth_type ipv4
  ip_proto sctp
  dst_port 80
  skip_sw
  in_hw
...
```

Future Enhancements

As described earlier the TC flower classifier already allows matching on a wide range of layer 2 through 4 header fields and tunnel metadata. There does, however, appear to be some scope to extend match coverage to include IPv6 Neighbour discovery and label and Geneve options.

There also appears to extend existing support for exact-match on fields of MPLS label stack entries to allow masked matches.

Lastly, while stateless matching is well supported there is currently no support for matching on stateful flow information. It seems that in conjunction with an appropriate action flower could support matching on connection state determined by conntrack.

Such a scheme could take inspiration from the conntrack support implemented in the Open vSwitch Linux kernel datapath whereby packets are passed to a conntrack action. After processing by that action a packet-classification occurs for a second time and may match on conntrack states. Such a scheme would allow enhanced security policies to be described by TC flower rules.

Conclusion

The TC flower classifier when used in conjunction with TC actions provides a rich mechanism to allow the construction of match/action datapaths with one or more table. The Linux kernel provides both a software implementation of this datapath and a control plane that may be used to configure both this software implementation and hardware implementations.

On the matching side this mechanism implements stateless matching on a variety of packet fields as well as tunnel metadata. And on the action side a variety of operations are supported including drop, output with and without cloning of the packet, and modification of packet fields and tunnel metadata.

This mechanism seems well suited being enhanced to cover more packet header fields and stateful information provided by conntrack.

Bibliography

References

[1] Fastabend, J. 2011a. net: implement mechanism for hw based qos. git.kernel.org. commit id: 4f57c087de9b.

[2] Fastabend, J. 2011b. net: sched: add cls_u32 offload hooks for netdevs. git.kernel.org. commit id: a1b7c5fd7fe9.