Devlink Trap

Background

Devices capable of offloading the kernel's datapath and perform functions such as bridging and routing must also be able to send specific packets to the kernel (i.e., the CPU) for processing.

For example, a device acting as a multicast-aware bridge must be able to send IGMP membership reports to the kernel for processing by the bridge module. Without processing such packets, the bridge module could never populate its MDB.

As another example, consider a device acting as router which has received an IP packet with a TTL of 1. Upon routing the packet the device must send it to the kernel so that it will route it as well and generate an ICMP Time Exceeded error datagram. Without letting the kernel route such packets itself, utilities such as `traceroute` could never work.

The fundamental ability of sending certain packets to the kernel for processing is called "packet trapping".

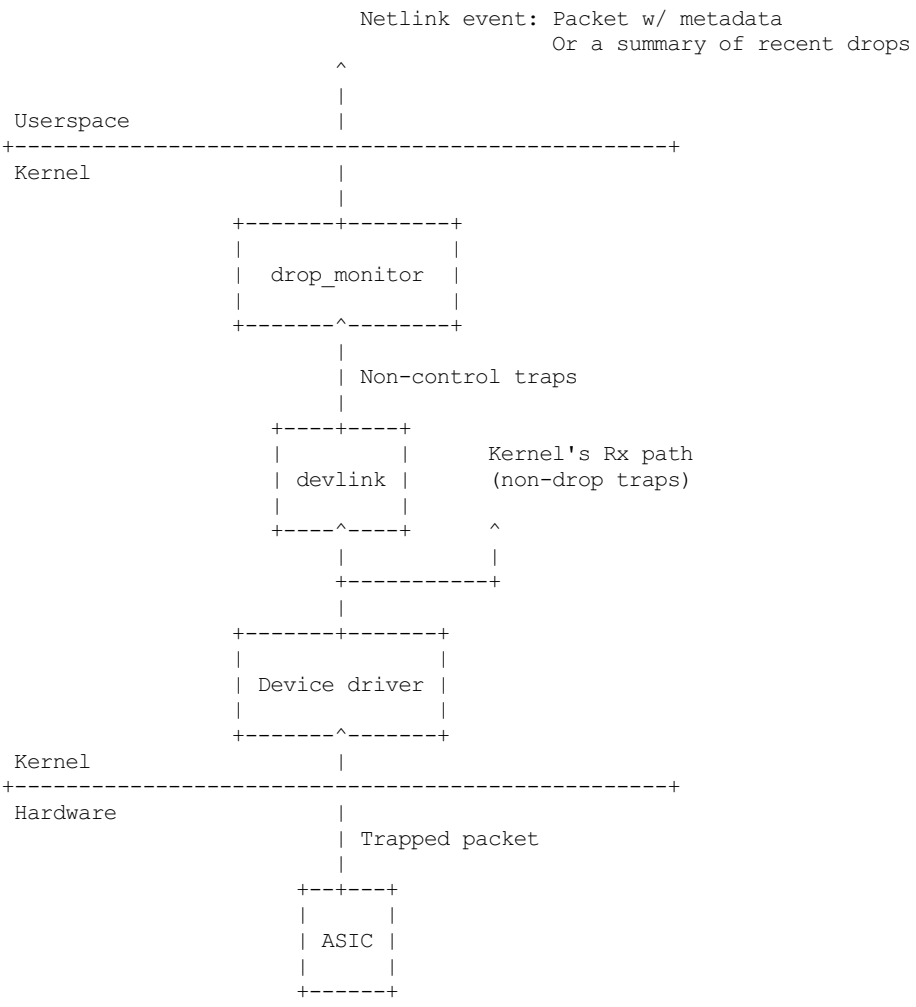
Overview

The `devlink-trap` mechanism allows capable device drivers to register their supported packet traps with `devlink` and report trapped packets to `devlink` for further analysis.

Upon receiving trapped packets, `devlink` will perform a per-trap packets and bytes accounting and potentially report the packet to user space via a netlink event along with all the provided metadata (e.g., trap reason, timestamp, input port). This is especially useful for drop traps (see :ref:`Trap-Types`) as it allows users to obtain further visibility into packet drops that would otherwise be invisible.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\networking\devlink\[linux-master][Documentation][networking][devlink]devlink-trap.rst, line 35); *backlink***
>
> Unknown interpreted text role "ref".

The following diagram provides a general overview of `devlink-trap`:



Trap Types

The `devlink-trap` mechanism supports the following packet trap types:

- `drop`: Trapped packets were dropped by the underlying device. Packets are only processed by `devlink` and not injected to the kernel's Rx path. The trap action (see :ref:`Trap-Actions`) can be changed.

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\networking\devlink\[linux-master][Documentation][networking][devlink]devlink-trap.rst`, line 91); *backlink*
  >
  > Unknown interpreted text role "ref".

- `exception`: Trapped packets were not forwarded as intended by the underlying device due to an exception (e.g., TTL error, missing neighbour entry) and trapped to the control plane for resolution. Packets are processed by `devlink` and injected to the kernel's Rx path. Changing the action of such traps is not allowed, as it can easily break the control plane.
- `control`: Trapped packets were trapped by the device because these are control packets required for the correct functioning of the control plane. For example, ARP request and IGMP query packets. Packets are injected to the kernel's Rx path, but not reported to the kernel's drop monitor. Changing the action of such traps is not allowed, as it can easily break the control plane.

Trap Actions

The `devlink-trap` mechanism supports the following packet trap actions:

- `trap`: The sole copy of the packet is sent to the CPU.
- `drop`: The packet is dropped by the underlying device and a copy is not sent to the CPU.
- `mirror`: The packet is forwarded by the underlying device and a copy is sent to the CPU.

Generic Packet Traps

Generic packet traps are used to describe traps that trap well-defined packets or packets that are trapped due to well-defined conditions (e.g., TTL error). Such traps can be shared by multiple device drivers and their description must be added to the following table:

List of Generic Packet Traps

Name	Type	Description
`source_mac_is_multicast`	`drop`	Traps incoming packets that the device decided to drop because of a multicast source MAC
`vlan_tag_mismatch`	`drop`	Traps incoming packets that the device decided to drop in case of VLAN tag mismatch: The ingress bridge port is not configured with a PVID and the packet is untagged or prio-tagged
`ingress_vlan_filter`	`drop`	Traps incoming packets that the device decided to drop in case they are tagged with a VLAN that is not configured on the ingress bridge port
`ingress_spanning_tree_filter`	`drop`	Traps incoming packets that the device decided to drop in case the STP state of the ingress bridge port is not "forwarding"
`port_list_is_empty`	`drop`	Traps packets that the device decided to drop in case they need to be flooded (e.g., unknown unicast, unregistered multicast) and there are no ports the packets should be flooded to
`port_loopback_filter`	`drop`	Traps packets that the device decided to drop in case after layer 2 forwarding the only port from which they should be transmitted through is the port from which they were received
`blackhole_route`	`drop`	Traps packets that the device decided to drop in case they hit a blackhole route
`ttl_value_is_too_small`	`exception`	Traps unicast packets that should be forwarded by the device whose TTL was decremented to 0 or less
`tail_drop`	`drop`	Traps packets that the device decided to drop because they could not be enqueued to a transmission queue which is full
`non_ip`	`drop`	Traps packets that the device decided to drop because they need to undergo a layer 3 lookup, but are not IP or MPLS packets
`uc_dip_over_mc_dmac`	`drop`	Traps packets that the device decided to drop because they need to be routed and they have a unicast destination IP and a multicast destination MAC

dip_is_loopback_address	drop	Traps packets that the device decided to drop because they need to be routed and their destination IP is the loopback address (i.e., 127.0.0.0/8 and ::1/128)
sip_is_mc	drop	Traps packets that the device decided to drop because they need to be routed and their source IP is multicast (i.e., 224.0.0.0/8 and ff::/8)
sip_is_loopback_address	drop	Traps packets that the device decided to drop because they need to be routed and their source IP is the loopback address (i.e., 127.0.0.0/8 and ::1/128)
ip_header_corrupted	drop	Traps packets that the device decided to drop because they need to be routed and their IP header is corrupted: wrong checksum, wrong IP version or too short Internet Header Length (IHL)
ipv4_sip_is_limited_bc	drop	Traps packets that the device decided to drop because they need to be routed and their source IP is limited broadcast (i.e., 255.255.255.255/32)
ipv6_mc_dip_reserved_scope	drop	Traps IPv6 packets that the device decided to drop because they need to be routed and their IPv6 multicast destination IP has a reserved scope (i.e., ffx0::/16)
ipv6_mc_dip_interface_local_scope	drop	Traps IPv6 packets that the device decided to drop because they need to be routed and their IPv6 multicast destination IP has an interface-local scope (i.e., ffx1::/16)
mtu_value_is_too_small	exception	Traps packets that should have been routed by the device, but were bigger than the MTU of the egress interface
unresolved_neigh	exception	Traps packets that did not have a matching IP neighbour after routing
mc_reverse_path_forwarding	exception	Traps multicast IP packets that failed reverse-path forwarding (RPF) check during multicast routing
reject_route	exception	Traps packets that hit reject routes (i.e., "unreachable", "prohibit")
ipv4_lpm_miss	exception	Traps unicast IPv4 packets that did not match any route
ipv6_lpm_miss	exception	Traps unicast IPv6 packets that did not match any route
non_routable_packet	drop	Traps packets that the device decided to drop because they are not supposed to be routed. For example, IGMP queries can be flooded by the device in layer 2 and reach the router. Such packets should not be routed and instead dropped
decap_error	exception	Traps NVE and IPinIP packets that the device decided to drop because of failure during decapsulation (e.g., packet being too short, reserved bits set in VXLAN header)
overlay_smac_is_mc	drop	Traps NVE packets that the device decided to drop because their overlay source MAC is multicast
ingress_flow_action_drop	drop	Traps packets dropped during processing of ingress flow action drop
egress_flow_action_drop	drop	Traps packets dropped during processing of egress flow action drop
stp	control	Traps STP packets
lacp	control	Traps LACP packets
lldp	control	Traps LLDP packets
igmp_query	control	Traps IGMP Membership Query packets
igmp_v1_report	control	Traps IGMP Version 1 Membership Report packets
igmp_v2_report	control	Traps IGMP Version 2 Membership Report packets
igmp_v3_report	control	Traps IGMP Version 3 Membership Report packets
igmp_v2_leave	control	Traps IGMP Version 2 Leave Group packets
mld_query	control	Traps MLD Multicast Listener Query packets
mld_v1_report	control	Traps MLD Version 1 Multicast Listener Report packets
mld_v2_report	control	Traps MLD Version 2 Multicast Listener Report packets
mld_v1_done	control	Traps MLD Version 1 Multicast Listener Done packets
ipv4_dhcp	control	Traps IPv4 DHCP packets
ipv6_dhcp	control	Traps IPv6 DHCP packets
arp_request	control	Traps ARP request packets
arp_response	control	Traps ARP response packets
arp_overlay	control	Traps NVE-decapsulated ARP packets that reached the overlay network. This is required, for example, when the address that needs to be resolved is a local address
ipv6_neigh_solicit	control	Traps IPv6 Neighbour Solicitation packets
ipv6_neigh_advert	control	Traps IPv6 Neighbour Advertisement packets
ipv4_bfd	control	Traps IPv4 BFD packets
ipv6_bfd	control	Traps IPv6 BFD packets

ipv4_ospf	control	Traps IPv4 OSPF packets
ipv6_ospf	control	Traps IPv6 OSPF packets
ipv4_bgp	control	Traps IPv4 BGP packets
ipv6_bgp	control	Traps IPv6 BGP packets
ipv4_vrrp	control	Traps IPv4 VRRP packets
ipv6_vrrp	control	Traps IPv6 VRRP packets
ipv4_pim	control	Traps IPv4 PIM packets
ipv6_pim	control	Traps IPv6 PIM packets
uc_loopback	control	Traps unicast packets that need to be routed through the same layer 3 interface from which they were received. Such packets are routed by the kernel, but also cause it to potentially generate ICMP redirect packets
local_route	control	Traps unicast packets that hit a local route and need to be locally delivered
external_route	control	Traps packets that should be routed through an external interface (e.g., management interface) that does not belong to the same device (e.g., switch ASIC) as the ingress interface
ipv6_uc_dip_link_local_scope	control	Traps unicast IPv6 packets that need to be routed and have a destination IP address with a link-local scope (i.e., fe80::/10). The trap allows device drivers to avoid programming link-local routes, but still receive packets for local delivery
ipv6_dip_all_nodes	control	Traps IPv6 packets that their destination IP address is the "All Nodes Address" (i.e., ff02::1)
ipv6_dip_all_routers	control	Traps IPv6 packets that their destination IP address is the "All Routers Address" (i.e., ff02::2)
ipv6_router_solicit	control	Traps IPv6 Router Solicitation packets
ipv6_router_advert	control	Traps IPv6 Router Advertisement packets
ipv6_redirect	control	Traps IPv6 Redirect Message packets
ipv4_router_alert	control	Traps IPv4 packets that need to be routed and include the Router Alert option. Such packets need to be locally delivered to raw sockets that have the IP_ROUTER_ALERT socket option set
ipv6_router_alert	control	Traps IPv6 packets that need to be routed and include the Router Alert option in their Hop-by-Hop extension header. Such packets need to be locally delivered to raw sockets that have the IPV6_ROUTER_ALERT socket option set
ptp_event	control	Traps PTP time-critical event messages (Sync, Delay_req, Pdelay_Req and Pdelay_Resp)
ptp_general	control	Traps PTP general messages (Announce, Follow_Up, Delay_Resp, Pdelay_Resp_Follow_Up, management and signaling)
flow_action_sample	control	Traps packets sampled during processing of flow action sample (e.g., via tc's sample action)
flow_action_trap	control	Traps packets logged during processing of flow action trap (e.g., via tc's trap action)
early_drop	drop	Traps packets dropped due to the RED (Random Early Detection) algorithm (i.e., early drops)
vxlan_parsing	drop	Traps packets dropped due to an error in the VXLAN header parsing which might be because of packet truncation or the I flag is not set.
llc_snap_parsing	drop	Traps packets dropped due to an error in the LLC+SNAP header parsing
vlan_parsing	drop	Traps packets dropped due to an error in the VLAN header parsing. Could include unexpected packet truncation.
pppoe_ppp_parsing	drop	Traps packets dropped due to an error in the PPPoE+PPP header parsing. This could include finding a session ID of 0xFFFF (which is reserved and not for use), a PPPoE length which is larger than the frame received or any common error on this type of header
mpls_parsing	drop	Traps packets dropped due to an error in the MPLS header parsing which could include unexpected header truncation
arp_parsing	drop	Traps packets dropped due to an error in the ARP header parsing
ip_1_parsing	drop	Traps packets dropped due to an error in the first IP header parsing. This packet trap could include packets which do not pass an IP checksum check, a header length check (a minimum of 20 bytes), which might suffer from packet truncation thus the total length field exceeds the received packet length etc

ip_n_parsing	drop	Traps packets dropped due to an error in the parsing of the last IP header (the inner one in case of an IP over IP tunnel). The same common error checking is performed here as for the ip_1_parsing trap
gre_parsing	drop	Traps packets dropped due to an error in the GRE header parsing
udp_parsing	drop	Traps packets dropped due to an error in the UDP header parsing. This packet trap could include checksum errorrs, an improper UDP length detected (smaller than 8 bytes) or detection of header truncation.
tcp_parsing	drop	Traps packets dropped due to an error in the TCP header parsing. This could include TCP checksum errors, improper combination of SYN, FIN and/or RESET etc.
ipsec_parsing	drop	Traps packets dropped due to an error in the IPSEC header parsing
sctp_parsing	drop	Traps packets dropped due to an error in the SCTP header parsing. This would mean that port number 0 was used or that the header is truncated.
dccp_parsing	drop	Traps packets dropped due to an error in the DCCP header parsing
gtp_parsing	drop	Traps packets dropped due to an error in the GTP header parsing
esp_parsing	drop	Traps packets dropped due to an error in the ESP header parsing
blackhole_nexthop	drop	Traps packets that the device decided to drop in case they hit a blackhole nexthop
dmac_filter	drop	Traps incoming packets that the device decided to drop because the destination MAC is not configured in the MAC table and the interface is not in promiscuous mode

Driver-specific Packet Traps

Device drivers can register driver-specific packet traps, but these must be clearly documented. Such traps can correspond to device-specific exceptions and help debug packet drops caused by these exceptions. The following list includes links to the description of driver-specific traps registered by various device drivers:

- Documentation/networking/devlink/netdevsim.rst
- Documentation/networking/devlink/mlxsw.rst
- Documentation/networking/devlink/prestera.rst

Generic Packet Trap Groups

Generic packet trap groups are used to aggregate logically related packet traps. These groups allow the user to batch operations such as setting the trap action of all member traps. In addition, `devlink-trap` can report aggregated per-group packets and bytes statistics, in case per-trap statistics are too narrow. The description of these groups must be added to the following table:

List of Generic Packet Trap Groups

Name	Description
l2_drops	Contains packet traps for packets that were dropped by the device during layer 2 forwarding (i.e., bridge)
l3_drops	Contains packet traps for packets that were dropped by the device during layer 3 forwarding
l3_exceptions	Contains packet traps for packets that hit an exception (e.g., TTL error) during layer 3 forwarding
buffer_drops	Contains packet traps for packets that were dropped by the device due to an enqueue decision
tunnel_drops	Contains packet traps for packets that were dropped by the device during tunnel encapsulation / decapsulation
acl_drops	Contains packet traps for packets that were dropped by the device during ACL processing
stp	Contains packet traps for STP packets
lacp	Contains packet traps for LACP packets
lldp	Contains packet traps for LLDP packets
mc_snooping	Contains packet traps for IGMP and MLD packets required for multicast snooping
dhcp	Contains packet traps for DHCP packets
neigh_discovery	Contains packet traps for neighbour discovery packets (e.g., ARP, IPv6 ND)
bfd	Contains packet traps for BFD packets
ospf	Contains packet traps for OSPF packets
bgp	Contains packet traps for BGP packets
vrrp	Contains packet traps for VRRP packets
pim	Contains packet traps for PIM packets
uc_loopback	Contains a packet trap for unicast loopback packets (i.e., `uc_loopback`). This trap is singled-out because in cases such as one-armed router it will be constantly triggered. To limit the impact on the CPU usage, a packet trap policer with a low rate can be bound to the group without affecting other traps
local_delivery	Contains packet traps for packets that should be locally delivered after routing, but do not match more specific packet traps (e.g., `ipv4_bgp`)

external_delivery	Contains packet traps for packets that should be routed through an external interface (e.g., management interface) that does not belong to the same device (e.g., switch ASIC) as the ingress interface
ipv6	Contains packet traps for various IPv6 control packets (e.g., Router Advertisements)
ptp_event	Contains packet traps for PTP time-critical event messages (Sync, Delay_req, Pdelay_Req and Pdelay_Resp)
ptp_general	Contains packet traps for PTP general messages (Announce, Follow_Up, Delay_Resp, Pdelay_Resp_Follow_Up, management and signaling)
acl_sample	Contains packet traps for packets that were sampled by the device during ACL processing
acl_trap	Contains packet traps for packets that were trapped (logged) by the device during ACL processing
parser_error_drops	Contains packet traps for packets that were marked by the device during parsing as erroneous

Packet Trap Policers

As previously explained, the underlying device can trap certain packets to the CPU for processing. In most cases, the underlying device is capable of handling packet rates that are several orders of magnitude higher compared to those that can be handled by the CPU.

Therefore, in order to prevent the underlying device from overwhelming the CPU, devices usually include packet trap policers that are able to police the trapped packets to rates that can be handled by the CPU.

The `devlink-trap` mechanism allows capable device drivers to register their supported packet trap policers with `devlink`. The device driver can choose to associate these policers with supported packet trap groups (see :ref:`Generic-Packet-Trap-Groups`) during its initialization, thereby exposing its default control plane policy to user space.

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\networking\devlink\[linux-master][Documentation][networking][devlink]devlink-trap.rst`, line 605); *backlink***
>
> Unknown interpreted text role "ref".

Device drivers should allow user space to change the parameters of the policers (e.g., rate, burst size) as well as the association between the policers and trap groups by implementing the relevant callbacks.

If possible, device drivers should implement a callback that allows user space to retrieve the number of packets that were dropped by the policer because its configured policy was violated.

Testing

See `tools/testing/selftests/drivers/net/netdevsim/devlink_trap.sh` for a test covering the core infrastructure. Test cases should be added for any new functionality.

Device drivers should focus their tests on device-specific functionality, such as the triggering of supported packet traps.