NAME

ibv_create_flow, ibv_destroy_flow − create or destroy flow steering rules

SYNOPSIS

#include <infiniband/verbs.h>

struct ibv_flow *ibv_create_flow(struct ibv_qp **qp*,
                    struct ibv_flow_attr **flow_attr*);
int ibv_destroy_flow(struct ibv_flow **flow_id*);

DESCRIPTION

ibv_create_flow()

allows a user application QP *qp* to be attached into a specified flow *flow* which is defined in *<infiniband/verbs.h>*

```
struct ibv_flow_attr {
                uint32_t comp_mask;             /* Future extendibility */
                enum ibv_flow_attr_type type;         /* Rule type - see below */
                uint16_t size;                  /* Size of command */
                uint16_t priority;              /* Rule priority - see below */
                uint8_t num_of_specs;           /* Number of ibv_flow_spec_xxx */
                uint8_t port;                   /* The uplink port number */
                uint32_t flags;                 /* Extra flags for rule - see below */
                /* Following are the optional layers according to user request
                 * struct ibv_flow_spec_xxx
                 * struct ibv_flow_spec_yyy
                 */
};

enum ibv_flow_attr_type {
                IBV_FLOW_ATTR_NORMAL            = 0x0,          /* Steering according to rule specificat
                IBV_FLOW_ATTR_ALL_DEFAULT       = 0x1,          /* Default unicast and multicast rule - r
                IBV_FLOW_ATTR_MC_DEFAULT        = 0x2,          /* Default multicast rule - receive all E
                IBV_FLOW_ATTR_SNIFFER           = 0x3,          /* Sniffer rule - receive all port traffic *
};

enum ibv_flow_flags {
                IBV_FLOW_ATTR_FLAGS_ALLOW_LOOP_BACK = 1 << 0,       /* Apply the rules on packets
                IBV_FLOW_ATTR_FLAGS_DONT_TRAP      = 1 << 1,        /* Rule doesn't trap received packets, a
                IBV_FLOW_ATTR_FLAGS_EGRESS         = 1 << 2,        /* Match sent packets against EGRESS
};

enum ibv_flow_spec_type {
                IBV_FLOW_SPEC_ETH               = 0x20,   /* Flow specification of L2 header */
                IBV_FLOW_SPEC_IPV4              = 0x30,   /* Flow specification of IPv4 header */
                IBV_FLOW_SPEC_IPV6              = 0x31,   /* Flow specification of IPv6 header */
                IBV_FLOW_SPEC_IPV4_EXT              = 0x32,   /* Extended flow specification of IPv4 */
                IBV_FLOW_SPEC_ESP              = 0x34,   /* Flow specification of ESP (IPSec) header */
                IBV_FLOW_SPEC_TCP              = 0x40,   /* Flow specification of TCP header */
                IBV_FLOW_SPEC_UDP               = 0x41,   /* Flow specification of UDP header */
                IBV_FLOW_SPEC_VXLAN_TUNNEL                  = 0x50,   /* Flow specification of VXLAN head
                IBV_FLOW_SPEC_GRE               = 0x51,   /* Flow specification of GRE header */
                IBV_FLOW_SPEC_MPLS                = 0x60,   /* Flow specification of MPLS header */
                IBV_FLOW_SPEC_INNER                  = 0x100,  /* Flag making L2/L3/L4 specifications to be a
                IBV_FLOW_SPEC_ACTION_TAG              = 0x1000, /* Action tagging matched packet */
                IBV_FLOW_SPEC_ACTION_DROP          = 0x1001, /* Action dropping matched packet */
                IBV_FLOW_SPEC_ACTION_HANDLE         = 0x1002, /* Carry out an action created by ibv_cre
                IBV_FLOW_SPEC_ACTION_COUNT          = 0x1003, /* Action count matched packet with a ibv
};
```

Flow specification general structure:

```
struct ibv_flow_spec_xxx {
                enum ibv_flow_spec_type  type;
                uint16_t  size;              /* Flow specification size = sizeof(struct ibv_flow_spec_xxx) */
                struct ibv_flow_xxx_filter val;
                struct ibv_flow_xxx_filter mask; /* Defines which bits from the filter value are applicable when lookin
};
```

Each spec struct holds the relevant network layer parameters for matching. To enforce the match, the user sets a mask for
Packets coming from the wire are matched against the flow specification. If a match is found, the associated flow actions a
In ingress flows, the QP parameter is treated as another action of scattering the packet to the respected QP.
If the bit is set in the mask, the corresponding bit in the value should be matched.
Note that most vendors support either full mask (all "1"s) or zero mask (all "0"s).
Network parameters in the relevant network structs should be given in network order (big endian).

Flow domains and priority

Flow steering defines the concept of domain and priority. Each domain represents an application that can attach a flow. Domains are prioritized. A higher priority domain will always supersede a lower priority domain when their flow specifications overlap.
IB verbs have the higher priority domain.
In addition to the domain, there is priority within each of the domains. A lower priority numeric value (higher priority) takes precedence over matching rules with higher numeric priority value (lower priority). It is important to note that the priority value of a flow spec is used not only to establish the precedence of conflicting flow matches but also as a way to abstract the order on which flow specs are tested for matches. Flows with higher priorities will be tested before flows with lower priorities.

Rules definition ordering

An application can provide the ibv_flow_spec_xxx rules in an un-ordered scheme. In this case, each spec should be well defined and match a specific network header layer. In some cases, when certain flow spec types are present in the spec list, it is required to provide the list in an ordered manner so that the position of that flow spec type in the protocol stack is strictly defined. When the certain spec type, which requires the ordering, resides in the inner network protocol stack (in tunnel protocols) the ordering should be applied to the inner network specs and should be combined with the inner spec indication using the IBV_FLOW_SPEC_INNER flag. For example: An MPLS spec which attempts to match an MPLS tag in the inner network should have the IBV_FLOW_SPEC_INNER flag set and so do the rest of the inner network specs. On top of that, all the inner network specs should be provided in an ordered manner. This is essential to represent many of the encapsulation tunnel protocols.

The flow spec types which require this sort of ordering are:
1. IBV_FLOW_SPEC_MPLS -
Since MPLS header can appear at several locations in the protocol stack and can also be encapsulated on top of different layers, it is required to place this spec according to its exact location in the protocol stack.

ibv_destroy_flow()

destroys the flow *flow_id*.

RETURN VALUE

ibv_create_flow() returns a pointer to the flow, or NULL if the request fails. In case of an error, errno is updated.

ibv_destroy_flow() returns 0 on success, or the value of errno on failure (which indicates the failure reason).

ERRORS

EINVAL

ibv_create_flow() flow specification, QP or priority are invalid

ibv_destroy_flow() flow_id is invalid

ENOMEM

Couldn't create/destroy flow, not enough memory

ENXIO

Device managed flow steering isn't currently supported

EPERM

No permissions to add the flow steering rule

NOTES

1. These verbs are available only for devices supporting
 IBV_DEVICE_MANAGED_FLOW_STEERING and only for QPs of Transport Service Type **IBV_QPT_UD** or **IBV_QPT_RAW_PACKET**
2. User must memset the spec struct with zeros before using it.
3. ether_type field in ibv_flow_eth_filter is the ethertype following the last VLAN tag of the packet.
4. Only rule type IBV_FLOW_ATTR_NORMAL supports IBV_FLOW_ATTR_FLAGS_DONT_TRAP flag.
5. No specifications are needed for IBV_FLOW_ATTR_SNIFFER rule type.
6. When IBV_FLOW_ATTR_FLAGS_EGRESS flag is set, the qp parameter is used only as a mean to get the device.

EXAMPLE

Below flow_attr defines a rule in priority 0 to match a destination mac address and a source ipv4 address. For that, L2 and L3 specs are used.
If there is a hit on this rule, means the received packet has destination mac: 66:11:22:33:44:55 and source ip: 0x0B86C806, the packet is steered to its attached qp.

```
struct raw_eth_flow_attr {
                struct ibv_flow_attr        attr;
                struct ibv_flow_spec_eth        spec_eth;
                struct ibv_flow_spec_ipv4       spec_ipv4;
} __attribute__((packed));

struct raw_eth_flow_attr flow_attr = {
                    .attr = {
                        .comp_mask      = 0,
                        .type           = IBV_FLOW_ATTR_NORMAL,
                        .size           = sizeof(flow_attr),
                        .priority       = 0,
                        .num_of_specs   = 2,
                        .port           = 1,
                        .flags          = 0,
                    },
                    .spec_eth = {
                        .type   = IBV_FLOW_SPEC_ETH,
                        .size   = sizeof(struct ibv_flow_spec_eth),
                        .val = {
                            .dst_mac = {0x66, 0x11, 0x22, 0x33, 0x44, 0x55},
                            .src_mac = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
                            .ether_type = 0,
                            .vlan_tag = 0,
                        },
                        .mask = {
                            .dst_mac = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                            .src_mac = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                            .ether_type = 0,
                            .vlan_tag = 0,
                        }
                    },
                    .spec_ipv4 = {
                        .type   = IBV_FLOW_SPEC_IPV4,
                        .size   = sizeof(struct ibv_flow_spec_ipv4),
                        .val = {
                            .src_ip = 0x0B86C806,
                            .dst_ip = 0,
                        },
                        .mask = {
                            .src_ip = 0xFFFFFFFF,
                            .dst_ip = 0,
                        }
                    }
};
```

AUTHORS

Hadar Hen Zion <hadarh@mellanox.com>

Matan Barak <matanb@mellanox.com>

Yishai Hadas <yishaih@mellanox.com>

Maor Gottlieb <maorg@mellanox.com>