FCI API Reference

Contents

1 Revision History 9

2 Module Index 11

2.1 Modules 11

3 Data Structure Index 12

3.1 Data Structures 12

4 File Index 14

4.1 File List 14

5 Module Documentation 15

5.1 LibFCI 15

5.1.1 Introduction 15

5.1.2 Acronyms and Definitions 16

5.1.3 Functions Summary 16

5.1.4 Commands Summary 17

5.1.5 Events summary 17

5.1.6 Interface Management 18

5.1.7 Features 20

5.1.7.1 IPv4/IPv6 Router (TCP/UDP) 20

5.1.7.2 L2 Bridge (Switch) 22

5.1.7.3 Flexible Parser 24

5.1.7.4 Flexible Router 25

5.1.8 Defines 29

5.1.8.1 FPP_CMD_PHY_IF 29

5.1.8.2 FPP_CMD_LOG_IF 30

5.1.8.3 FPP_CMD_IF_LOCK_SESSION 32

5.1.8.4 FPP_CMD_IF_UNLOCK_SESSION 33

5.1.8.5 FPP_CMD_L2_BD 33

5.1.8.6 FPP_CMD_FP_TABLE . . . 36
5.1.8.7 FPP_CMD_FP_RULE . . . 38
5.1.8.8 FPP_CMD_FP_FLEXIBLE_FILTER . . . 39
5.1.8.9 FPP_CMD_DATA_BUF_PUT . . . 40
5.1.8.10 FPP_CMD_DATA_BUF_AVAIL . . . 41
5.1.8.11 FCI_CFG_FORCE_LEGACY_API . . . 41
5.1.8.12 FPP_CMD_IPV4_CONNTRACK_CHANGE . . . 41
5.1.8.13 FPP_CMD_IPV6_CONNTRACK_CHANGE . . . 41
5.1.8.14 CTCMD_FLAGS_REP_DISABLED . . . 42
5.1.9 Enums . . . 42
5.1.9.1 fpp_if_flags_t . . . 42
5.1.9.2 fpp_phy_if_op_mode_t . . . 42
5.1.9.3 fpp_if_m_rules_t . . . 43
5.1.9.4 fpp_phy_if_block_state_t . . . 43
5.1.9.5 fpp_l2_bd_flags_t . . . 44
5.1.9.6 fpp_fp_rule_match_action_t . . . 44
5.1.9.7 fpp_fp_offset_from_t . . . 44
5.1.9.8 fci_mcast_groups_t . . . 45
5.1.9.9 fci_client_type_t . . . 45
5.1.9.10 fci_cb_retval_t . . . 45
5.1.10 Functions . . . 46
5.1.10.1 fci_open() . . . 46
5.1.10.2 fci_close() . . . 46
5.1.10.3 fci_catch() . . . 47
5.1.10.4 fci_cmd() . . . 47
5.1.10.5 fci_query() . . . 48
5.1.10.6 fci_write() . . . 49
5.1.10.7 fci_register_cb() . . . 50

6 Data Structure Documentation **51**
6.1 FCI_CLIENT Struct Reference . . . 51
6.1.1 Detailed Description . . . 51
6.2 fpp_algo_stats_t Struct Reference . . . 51
6.2.1 Detailed Description . . . 52
6.2.2 Field Documentation . . . 52
6.2.2.1 processed . . . 52

6.2.2.2 accepted 52
6.2.2.3 rejected 52
6.2.2.4 discarded 52
6.3 fpp_buf_cmd_t Struct Reference 52
6.3.1 Detailed Description 53
6.3.2 Field Documentation 53
6.3.2.1 payload 53
6.3.2.2 len 53
6.4 fpp_ct6_cmd_t Struct Reference 53
6.4.1 Detailed Description 54
6.4.2 Field Documentation 54
6.4.2.1 action 54
6.4.2.2 saddr 54
6.4.2.3 daddr 54
6.4.2.4 sport 54
6.4.2.5 dport 55
6.4.2.6 saddr_reply 55
6.4.2.7 daddr_reply 55
6.4.2.8 sport_reply 55
6.4.2.9 dport_reply 55
6.4.2.10 protocol 55
6.4.2.11 flags 55
6.4.2.12 route_id 55
6.4.2.13 route_id_reply 56
6.5 fpp_ct_cmd_t Struct Reference 56
6.5.1 Detailed Description 56
6.5.2 Field Documentation 56
6.5.2.1 action 57
6.5.2.2 saddr 57
6.5.2.3 daddr 57
6.5.2.4 sport 57
6.5.2.5 dport 57
6.5.2.6 saddr_reply 57
6.5.2.7 daddr_reply 57
6.5.2.8 sport_reply 58
6.5.2.9 dport_reply 58

6.5.2.10 protocol 58
6.5.2.11 flags 58
6.5.2.12 route_id 58
6.5.2.13 route_id_reply 58
6.6 fpp_fp_rule_cmd_t Struct Reference 58
6.6.1 Detailed Description 59
6.6.2 Field Documentation 59
6.6.2.1 action 59
6.6.2.2 r 59
6.7 fpp_fp_rule_props_t Struct Reference 59
6.7.1 Detailed Description 59
6.8 fpp_fp_table_cmd_t Struct Reference 60
6.8.1 Detailed Description 60
6.8.2 Field Documentation 60
6.8.2.1 action 60
6.8.2.2 table_name 60
6.8.2.3 rule_name 60
6.8.2.4 position 60
6.8.2.5 r 61
6.9 fpp_if_m_args_t Struct Reference 61
6.9.1 Detailed Description 61
6.9.2 Field Documentation 61
6.9.2.1 vlan 62
6.9.2.2 ethtype 62
6.9.2.3 sport 62
6.9.2.4 dport 62
6.9.2.5 v4 62
6.9.2.6 v6 62
6.9.2.7 proto 62
6.9.2.8 smac 63
6.9.2.9 dmac 63
6.9.2.10 fp_table0 63
6.9.2.11 fp_table1 63
6.9.2.12 hif_cookie 63
6.10 fpp_l2_bd_cmd_t Struct Reference 63
6.10.1 Detailed Description 64

6.10.2 Field Documentation 64
6.10.2.1 action 64
6.10.2.2 vlan 64
6.10.2.3 ucast_hit 64
6.10.2.4 ucast_miss 64
6.10.2.5 mcast_hit 64
6.10.2.6 mcast_miss 65
6.10.2.7 if_list 65
6.10.2.8 untag_if_list 65
6.10.2.9 flags 65
6.11 fpp_log_if_cmd_t Struct Reference 65
6.11.1 Detailed Description 66
6.11.2 Field Documentation 66
6.11.2.1 action 66
6.11.2.2 name 66
6.11.2.3 id 67
6.11.2.4 parent_name 67
6.11.2.5 parent_id 67
6.11.2.6 egress 67
6.11.2.7 flags 67
6.11.2.8 match 68
6.11.2.9 arguments 68
6.11.2.10 stats 68
6.12 fpp_phy_if_cmd_t Struct Reference 68
6.12.1 Detailed Description 69
6.12.2 Field Documentation 69
6.12.2.1 action 69
6.12.2.2 name 69
6.12.2.3 id 69
6.12.2.4 flags 70
6.12.2.5 mode 70
6.12.2.6 block_state 70
6.12.2.7 mac_addr 70
6.12.2.8 mirror 70
6.12.2.9 stats 70
6.13 fpp_phy_if_stats_t Struct Reference 71

6.13.1 Detailed Description . . . 71
6.13.2 Field Documentation . . . 71
6.13.2.1 ingress . . . 71
6.13.2.2 egress . . . 71
6.13.2.3 malformed . . . 71
6.13.2.4 discarded . . . 72
6.14 fpp_rt_cmd_t Struct Reference . . . 72
6.14.1 Detailed Description . . . 72
6.14.2 Field Documentation . . . 72
6.14.2.1 action . . . 73
6.14.2.2 dst_mac . . . 73
6.14.2.3 output_device . . . 73
6.14.2.4 id . . . 73
6.14.2.5 flags . . . 73
6.15 fpp_timeout_cmd_t Struct Reference . . . 74
6.15.1 Detailed Description . . . 74

7 File Documentation **75**
7.1 fpp.h File Reference . . . 75
7.1.1 Detailed Description . . . 76
7.1.2 Macro Definition Documentation . . . 76
7.1.2.1 FPP_CMD_IPV4_CONNTRACK . . . 76
7.1.2.2 FPP_CMD_IPV6_CONNTRACK . . . 78
7.1.2.3 FPP_CMD_IP_ROUTE . . . 81
7.1.2.4 FPP_CMD_IPV4_RESET . . . 82
7.1.2.5 FPP_CMD_IPV6_RESET . . . 82
7.1.2.6 FPP_CMD_IPV4_SET_TIMEOUT . . . 82
7.2 fpp_ext.h File Reference . . . 83
7.2.1 Detailed Description . . . 85
7.3 libfci.h File Reference . . . 85
7.3.1 Detailed Description . . . 87

8 Example Documentation **88**
8.1 fpp_cmd_ip_route.c . . . 88
8.2 fpp_cmd_ipv4_conntrack.c . . . 91
8.3 fpp_cmd_ipv6_conntrack.c . . . 96
8.4 fpp_cmd_log_if.c . . . 100

8.5 fpp_cmd_phy_if.c 102

Index **107**

Chapter 1

Revision History

Revision	Change Description
1.0.0	Initial version. Contains description of FCI API and following features: • Interface Management • IPv4/IPv6 Router (TCP/UDP) • L2 Bridge (Switch) • Flexible Parser • Flexible Router
1.1.0	Added description of simple bridge (without VLAN awareness). Disabled part describing async messaging as it is currently not used. Description of fci_cmd(), fci_query(), and fci_write() simplified. Various minor improvements.
1.2.0	Improved description of Router and Bridge configuration steps. Added missing byte order information to various command argument values. Following values unified with rest of structure members to be in network byte order: • `fpp_rt_cmd_t::id` • `fpp_rt_cmd_t::flags` • `fpp_ct_cmd_t::route_id` • `fpp_ct_cmd_t::route_id_reply` • `fpp_ct_cmd_t::flags` • `fpp_fp_table_cmd_t::position`
1.2.1	Added FPP_IF_MIRROR to fpp_if_flags_t. Added name of interface to mirror the traffic to fpp_phy_if_cmd_t.

1.3.0	Description of various elements re-phrased to better explain their purpose. Created summary lists of functions, commands, and events and added links to them to improve document navigation. Added usage examples for FPP_CMD_PHY_IF, FPP_CMD_LOG_IF, and FPP_CMD_IP_ROUTE commands. Described relevant fpp_rt_cmd_t structure members.
1.4.0	Added usage examples for FPP_CMD_IPV4_CONNTRACK and FPP_CMD_IPV6_CONNTRACK. Related argument structures documentation updated. Removed unwanted and unsupported symbol descriptions.
1.5.0	Added statistics for physical `fpp_phy_if_stats_t` and logical `fpp_algo_stats_t` interfaces . Statistics are in network byte order.
1.6.0	Added API for data passing: `FPP_CMD_DATA_BUF_PUT` and `FPP_CMD_DATA_BUF_AVAIL` with related `fpp_buf_cmd_t`.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

LibFCI 15

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

FCI_CLIENT
The FCI client representation type 51
fpp_algo_stats_t
Algorithm statistics 51
fpp_buf_cmd_t
Argument structure for the FPP_CMD_DATA_BUF_PUT command 52
fpp_ct6_cmd_t
Data structure used in various functions for IPv6 conntrack management . 53
fpp_ct_cmd_t
Data structure used in various functions for conntrack management 56
fpp_fp_rule_cmd_t
Arguments for the FPP_CMD_FP_RULE command 58
fpp_fp_rule_props_t
Properties of the Flexible parser rule 59
fpp_fp_table_cmd_t
Arguments for the FPP_CMD_FP_TABLE command 60
fpp_if_m_args_t
Match rules arguments 61
fpp_l2_bd_cmd_t
Data structure to be used for command buffer for L2 bridge domain control commands 63
fpp_log_if_cmd_t
Data structure to be used for logical interface commands 65
fpp_phy_if_cmd_t
Data structure to be used for physical interface commands 68
fpp_phy_if_stats_t
Physical interface statistics 71
fpp_rt_cmd_t
Structure representing the command to add or remove a route 72

fpp_timeout_cmd_t
Timeout command argument 74

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

fci_examples.h . . . . **??**
fpp.h
The legacy FCI API 75
fpp_ext.h
Extension of the legacy fpp.h 83
libfci.h
Generic LibFCI header file 85

Chapter 5

Module Documentation

5.1 LibFCI

This is the Fast Control Interface available to host applications to communicate with the networking engine.

5.1.1 Introduction

This is the Fast Control Interface available to host applications to communicate with the networking engine.

The FCI is intended to provide a generic configuration and monitoring interface to networking acceleration HW. Provided API shall remain the same within all HW/OS-specific implementations to keep dependent applications portable across various systems.

The LibFCI is not directly touching the HW. Instead, it only passes commands to dedicated software component (OS/HW-specific endpoint) and receives return values. The endpoint is then responsible for HW configuration. This approach supports kernel-user space deployment where user space contains only API and the logic is implemented in kernel.

Implementation uses appropriate transport mechanism to pass data between LibFCI user and the endpoint. For reference, in Linux netlink socket will be used, in QNX it will be a message.

Usage scenario example - FCI command execution:

1. User calls fci_open() to get the FCI_CLIENT instance, use FCI_GROUP_NONE as multicast group mask.
2. User calls fci_cmd() to send a command with arguments to the endpoint.
3. Endpoint receives the command and performs requested actions.
4. Endpoint generates response and sends it back to the client.
5. Client receives the response and informs the caller.
6. User calls fci_close() to finalize the FCI_CLIENT instance.

5.1.2 Acronyms and Definitions

- **Physical Interface:** Interface physically able to send and receive data (EMAC, HIF). Physical interfaces are pre-defined and can't be added or removed in runtime. Every physical interface has associated a **default** logical interface and set of properties like classification algorithm.

- **Logical Interface:** Extension of physical interface defined by set of rules which describes Ethernet traffic. Intended to be used to dispatch traffic being received via particular physical interfaces using 1:N association i.e. traffic received by a physical interface can be classified and distributed to N logical interfaces. These can be either connected to SW stack running in host system or just used to distribute traffic to an arbitrary physical interface(s). Logical interfaces are dynamic objects and can be created and destroyed in runtime.

- **Classification Algorithm:** Way how ingress traffic is being processed by the PFE firmware.

- **Route:** Routes are representing direction where matching traffic shall be forwarded to. Every route specifies egress physical interface and MAC address of next network node.

- **Conntrack:** "Tracked connection", a data structure containing information about a connection. In context of this document it always refers to an IP connection (TCP, UDP, other). Term is equal to 'routing table entry'. Conntracks contain reference to routes which shall be used in case when a packet is matching the conntrack properties.

5.1.3 Functions Summary

- fci_open()
  Connect to endpoint and create client instance.

- fci_close()
  Close connection and destroy the client instance.

- fci_write()
  Execute FCI command without data response.

- fci_cmd()
  Execute FCI command with data response.

- fci_query()
  Alternative to fci_cmd().

- fci_catch()
  Poll for and process received asynchronous messages.

- fci_register_cb()
  Register callback to be called in case of received message.

5.1.4 Commands Summary

- FPP_CMD_PHY_IF
  Management of physical interfaces.
- FPP_CMD_LOG_IF
  Management of logical interfaces.
- FPP_CMD_IF_LOCK_SESSION
  Get exclusive access to interfaces.
- FPP_CMD_IF_UNLOCK_SESSION
  Cancel exclusive access to interfaces.
- FPP_CMD_L2_BD
  L2 bridge domains management.
- FPP_CMD_FP_TABLE
  Administration of Flexible Parser tables.
- FPP_CMD_FP_RULE
  Administration of Flexible Parser rules.
- FPP_CMD_FP_FLEXIBLE_FILTER
  Utilization of Flexible Parser to filter out (drop) frames.
- FPP_CMD_IPV4_RESET
  Reset IPv4 (routes, conntracks, ...).
- FPP_CMD_IPV6_RESET
  Reset IPv6 (routes, conntracks, ...).
- FPP_CMD_IP_ROUTE
  Management of IP routes.
- FPP_CMD_IPV4_CONNTRACK
  Management of IPv4 connections.
- FPP_CMD_IPV6_CONNTRACK
  Management of IPv6 connections.
- FPP_CMD_IPV4_SET_TIMEOUT
  Configuration of connection timeouts.
- FPP_CMD_DATA_BUF_PUT
  Send arbitrary data to the accelerator.

5.1.5 Events summary

- FPP_CMD_DATA_BUF_AVAIL
  Network accelerator sends a data buffer to host.

5.1.6 Interface Management

Physical Interface

Physical interfaces are static objects and are defined at startup. LibFCI client can get a list of currently available physical interfaces using query option of the FPP_CMD_PHY_IF command. Every physical interface contains a list of logical interfaces. Without any configuration all physical interfaces are in default operation mode. It means that all ingress traffic is processed using only associated **default** logical interface. Default logical interface is always the tail of the list of associated logical interfaces. When new logical interface is associated, it is placed at head position of the list so the default one remains on tail. User can change the used classification algorithm via update option of the FPP_CMD_PHY_IF command.

Here are supported operations related to physical interfaces:

To **list** available physical interfaces:

1. Lock interface database with FPP_CMD_IF_LOCK_SESSION.
2. Read first interface via FPP_CMD_PHY_IF + FPP_ACTION_QUERY.
3. Read next interface(s) via FPP_CMD_PHY_IF + FPP_ACTION_QUERY_CONT.
4. Unlock interface database with FPP_CMD_IF_UNLOCK_SESSION.

To **modify** a physical interface (read-modify-write):

1. Lock interface database with FPP_CMD_IF_LOCK_SESSION.
2. Read interface properties via FPP_CMD_PHY_IF + FPP_ACTION_QUERY + FPP_ACTION_QUERY_CONT.
3. Modify desired properties.
4. Write modifications using FPP_CMD_PHY_IF + FPP_ACTION_UPDATE.
5. Unlock interface database with FPP_CMD_IF_UNLOCK_SESSION.

Logical Interface

Logical interfaces specify traffic endpoints. They are connected to respective physical interfaces and contain information about which traffic can they accept and how the accepted traffic shall be processed (where, resp. to which **physical** interface(s) the matching traffic shall be forwarded). For example, there can be two logical interfaces associated with an EMAC1, one accepting traffic with VLAN ID = 10 and the second one accepting all remaining traffic. First one can be configured to forward the matching traffic to EMAC1 and the second one to drop the rest.

Logical interfaces can be created and destroyed in runtime using actions related to the FPP_CMD_LOG_IF command. Note that first created logical interface on a physical interface becomes the default one (tail). All subsequent logical interfaces are added at head position of list of interfaces.

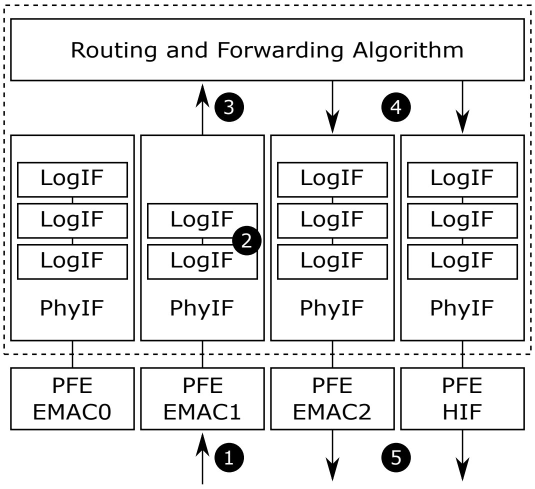

Figure 5.1 Configuration Example

The example shows scenario when physical interface EMAC1 is configured in FPP_IF_OP_FLEXIBLE_ROUTER operation mode:

1. Packet is received via EMAC1 port of the PFE.
2. Classifier walks through list of Logical Interfaces associated with the ingress Physical Interface. Every Logical Interface contains a set of classification rules (see fpp_if_m_rules_t) the classification process is using to match the ingress packet. Note that the list is searched from head to tail, where tail is the default logical interface.
3. Information about matching Logical Interface and Physical Interface is passed to the Routing and Forwarding Algorithm. The algorithm reads the Logical Interface and retrieves forwarding properties.
4. The matching Logical Interface is configured to forward the packet to EMAC2 and HIF so the forwarding algorithm ensures that. Optionally, here the packet can be modified according to interface setup (VLAN insertion, source MAC address replacement, ...).
5. Packet is physically transmitted via dedicated interfaces. Packet replica sent to HIF carries metadata describing the matching Logical and Physical interface so the host driver can easily dispatch the traffic.

Here are supported operations related to logical interfaces:

To **create** new logical interface:

1. Lock interface database with FPP_CMD_IF_LOCK_SESSION.
2. Create logical interface via FPP_CMD_LOG_IF + FPP_ACTION_REGISTER.
3. Unlock interface database with FPP_CMD_IF_UNLOCK_SESSION.

To **list** available logical interfaces:

1. Lock interface database with FPP_CMD_IF_LOCK_SESSION.

2. Read first interface via FPP_CMD_LOG_IF + FPP_ACTION_QUERY.
3. Read next interface(s) via FPP_CMD_LOG_IF + FPP_ACTION_QUERY_CONT.
4. Unlock interface database with FPP_CMD_IF_UNLOCK_SESSION.

To **modify** an interface (read-modify-write):

1. Lock interface database with FPP_CMD_IF_LOCK_SESSION.
2. Read interface properties via FPP_CMD_LOG_IF + FPP_ACTION_QUERY + FPP_ACTION_QUERY_CONT.
3. Modify desired properties.
4. Write modifications using FPP_CMD_LOG_IF + FPP_ACTION_UPDATE.
5. Unlock interface database with FPP_CMD_IF_UNLOCK_SESSION.

To **remove** logical interface:

1. Lock interface database with FPP_CMD_IF_LOCK_SESSION.
2. Remove logical interface via FPP_CMD_LOG_IF + FPP_ACTION_DEREGISTER.
3. Unlock interface database with FPP_CMD_IF_UNLOCK_SESSION.

5.1.7 Features

5.1.7.1 IPv4/IPv6 Router (TCP/UDP)

Introduction

The IPv4/IPv6 Forwarder is a dedicated feature to offload the host CPU from tasks related to forwarding of specific IP traffic between physical interfaces. Normally, the ingress IP traffic is passed to the host CPU running TCP/IP stack which is responsible for routing of the packets. Once the stack identifies that a packet does not belong to any of local IP endpoints it performs lookup in routing table to determine how to process such traffic. If the routing table contains entry associated with the packet (5-tuple search) the stack modifies and forwards the packet to another interface to reach its intended destination node. The PFE can be configured to identify flows which do not need to enter the host CPU using its internal routing table, and to ensure that the right packets are forwarded to the right destination interfaces.

Configuration

The FCI contains mechanisms to setup particular Physical Interfaces to start classifying packets using Router classification algorithm as well as to manage PFE routing tables. The router configuration then consists of following steps:

1. Optionally use FPP_CMD_IPV4_RESET or FPP_CMD_IPV6_RESET to initialize the router. All previous configuration changes will be discarded.

2. Create one or more routes (FPP_CMD_IP_ROUTE + FPP_ACTION_REGISTER). Once created, every route has an unique identifier. Creating route on an physical interface causes switch of operation mode of that interface to FPP_IF_OP_ROUTER.

3. Create one or more IPv4 routing table entries (FPP_CMD_IPV4_CONNTRACK + FPP_ACTION_REGISTER).

4. Create one or more IPv6 routing table entries (FPP_CMD_IPV6_CONNTRACK + FPP_ACTION_REGISTER).

5. Set desired physical interface(s) to router mode FPP_IF_OP_ROUTER using FPP_CMD_PHY_IF + FPP_ACTION_UPDATE. This selects interfaces which will use routing algorithm to classify ingress traffic.

6. Enable physical interface(s) by setting the FPP_IF_ENABLED flag via the FPP_CMD_PHY_IF + FPP_ACTION_UPDATE.

7. Optionally change MAC address(es) via FPP_CMD_PHY_IF + FPP_ACTION_UPDATE.

From this point the traffic matching created conntracks is processed according to conntrack properties (e.g. NAT) and fast-forwarded to configured physical interfaces. Conntracks are subject of aging. When no traffic has been seen for specified time period (see FPP_CMD_IPV4_SET_TIMEOUT) the conntracks are removed.

Routes and conntracks can be listed using query commands:

- FPP_CMD_IP_ROUTE + FPP_ACTION_QUERY + FPP_ACTION_QUERY_CONT.
- FPP_CMD_IPV4_CONNTRACK + FPP_ACTION_QUERY + FPP_ACTION_QUERY_CONT.
- FPP_CMD_IPV6_CONNTRACK + FPP_ACTION_QUERY + FPP_ACTION_QUERY_CONT.

When conntrack or route are no more required, they can be deleted via corresponding command:

- FPP_CMD_IP_ROUTE + FPP_ACTION_DEREGISTER,
- FPP_CMD_IPV4_CONNTRACK + FPP_ACTION_DEREGISTER, and
- FPP_CMD_IPV6_CONNTRACK + FPP_ACTION_DEREGISTER.

Deleting route causes deleting all associated conntracks. When the latest route on an interface is deleted, the interface is put to default operation mode FPP_IF_OP_DEFAULT.

5.1.7.2 L2 Bridge (Switch)

Introduction

The L2 Bridge functionality covers forwarding of packets based on MAC addresses. It provides possibility to move bridging-related tasks from host CPU to the PFE and thus offloads the host-based networking stack. The L2 Bridge feature represents a network switch device implementing following functionality:

- **MAC table and address learning:** The L2 bridging functionality is based on determining to which interface an ingress packet shall be forwarded. For this purpose a network switch device implements so called bridging table (MAC table) which is searched to get target interface for each packet entering the switch. If received source MAC address does not match any MAC table entry then a new entry, containing the Physical Interface which the packet has been received on, is added - learned. Destination MAC address of an ingress packet is then used to search the table to determine the target interface.
- **Aging:** Each MAC table entry gets default timeout value once learned. In time this timeout is being decreased until zero is reached. Entries with zero timeout value are automatically removed from the table. The timeout value is re-set each time the corresponding table entry is used to process a packet.
- **Port migration:** When a MAC address is seen on one interface of the switch and an entry has been created, it is automatically updated when the MAC address is seen on another interface.
- **VLAN Awareness:** The bridge implements VLAN table. This table is used to implement VLAN-based policies like Ingress and Egress port membership. Feature includes configurable VLAN tagging and un-tagging functionality per bridge interface (Physical Interface). The bridge utilizes PFE HW accelerators to perform MAC and VLAN table lookup thus this operation is highly optimized. Host CPU SW is only responsible for correct bridge configuration using the dedicated API.

L2 Bridge VLAN Awareness and Domains

The VLAN awareness is based on entities called Bridge Domains (BD) which are visible to both classifier firmware, and the driver, and are used to abstract particular VLANs. Every BD contains configurable set of properties:

- Associated VLAN ID.
- Set of Physical Interfaces which are members of the domain.
- Information about which of the member interfaces are 'tagged' or 'untagged'.
- Instruction how to process matching uni-cast packets (forward, flood, discard, ...).
- Instruction how to process matching multi-cast packets.

The L2 Bridge then consists of multiple BD types:

- **The Default BD:** Default domain is used by the classification process when a packet has been received with default VLAN ID. This can happen either if the packet does not contain VLAN tag or the VLAN tag is equal to the default VLAN configured within the bridge.
- **The Fall-back BD:** This domain is used when packet with an unknown VLAN ID (does not match any standard or default domain) is received in FPP_IF_OP_VLAN_BRIDGE mode. It is also used as representation of simple L2 bridge when VLAN awareness is disabled (in case of the FPP_IF_OP_BRIDGE mode).
- **Set of particular Standard BDs:** Standard domain. Specifies what to do when packet with VLAN ID matching the Standard BD is received.

Configuration

Here are steps needed to configure VLAN-aware switch:

1. Optionally get list of available physical interfaces and their IDs. See the Interface Management.
2. Create a bridge domain (VLAN domain) (FPP_CMD_L2_BD + FPP_ACTION_REGISTER).
3. Configure domain hit/miss actions (FPP_CMD_L2_BD + FPP_ACTION_UPDATE) to let the bridge know how to process matching traffic.
4. Add physical interfaces as members of that domain (FPP_CMD_L2_BD + FPP_ACTION_UPDATE). Adding interface to a bridge domain causes switch of its operation mode to FPP_IF_OP_VLAN_BRIDGE and enabling promiscuous mode on MAC level.
5. Set physical interface(s) to VLAN bridge mode FPP_IF_OP_VLAN_BRIDGE using FPP_CMD_PHY_IF + FPP_ACTION_UPDATE.
6. Set promiscuous mode and enable physical interface(s) by setting the FPP_IF_ENABLED and FPP_IF_PROMISC flags via the FPP_CMD_PHY_IF + FPP_ACTION_UPDATE.

For simple, non-VLAN aware switch do:

1. Optionally get list of available physical interfaces and their IDs. See the Interface Management.
2. Add physical interfaces as members of fall-back BD (FPP_CMD_L2_BD + FPP_ACTION_UPDATE). The fall-back BD is identified by VLAN 0 and exists automatically.
3. Configure domain hit/miss actions (FPP_CMD_L2_BD + FPP_ACTION_UPDATE) to let the bridge know how to process matching traffic.
4. Set physical interface(s) to simple bridge mode FPP_IF_OP_BRIDGE using FPP_CMD_PHY_IF + FPP_ACTION_UPDATE.

5. Set promiscuous mode and enable physical interface(s) by setting the FPP_IF_ENABLED and FPP_IF_PROMISC flags via the FPP_CMD_PHY_IF + FPP_ACTION_UPDATE.

Once interfaces are in bridge domain, all ingress traffic is processed according to bridge domain setup. Unknown source MAC addresses are being learned and after specified time period without traffic are being aged.

An interface can be added to or removed from BD at any time via FPP_CMD_L2_BD + FPP_ACTION_UPDATE. When interface is removed from all bridge domains (is not associated with any BD), its operation mode is automatically switched to FPP_IF_OP_DEFAULT and MAC promiscuous mode is disabled.

List of available bridge domains with their properties can be retrieved using FPP_CMD_L2_BD + FPP_ACTION_QUERY + FPP_ACTION_QUERY_CONT.

5.1.7.3 Flexible Parser

Introduction

The Flexible Parser is PFE firmware-based feature allowing user to extend standard ingress packet classification process by set of customizable classification rules. According to the rules the Flexible Parser can mark frames as ACCEPTED or REJECTED. The rules are configurable by user and exist in form of tables. Every classification table entry consist of following fields:

- 32-bit Data field to be compared with value from the ingress frame.
- 32-bit Mask field (active bits are '1') specifying which bits of the data field will be used to perform the comparison.
- 16-bit Configuration field specifying rule properties including the offset to the frame data which shall be compared.

The number of entries within the table is configurable by user. The table is processed sequentially starting from entry index 0 until the last one is reached or classification is terminated by a rule configuration. When none of rules has decided that the packet shall be accepted or rejected the default result is REJECT.

Example

This is example of how Flexible Parser table can be configured. Every row contains single rule and processing starts with rule 0. ACCEPT/REJECT means that the classification is terminated with given result, CONTINUE means that next rule (sequentially) will be evaluated. CONTINUE with N says that next rule to be evaluated is N. Evaluation of the latest rule not resulting in ACCEPT or REJECT results in REJECT.

Rule	Flags	Mask	Next	Condition
0	FP_FL_INVERT FP_FL_REJECT	!= 0	n/a	if ((PacketData&Mask) != (RuleData&Mask)) then REJECT else CONTINUE
1	FP_FL_ACCEPT	!= 0	n/a	if ((PacketData&Mask) == (RuleData&Mask)) then ACCEPT else CONTINUE
2	-	!= 0	4	if ((PacketData&Mask) == (RuleData&Mask)) then CONTINUE with 4 else CONTINUE
3	FP_FL_REJECT	= 0	n/a	REJECT
4	FP_FL_INVERT	!= 0	6	if ((PacketData&Mask) != (RuleData&Mask)) then CONTINUE with 6 else CONTINUE
5	FP_FL_ACCEPT	= 0	n/a	ACCEPT
6	FP_FL_INVERT FP_FL_ACCEPT	!= 0	n/a	if ((PacketData&Mask) != (RuleData&Mask)) then ACCEPT else CONTINUE
7	FP_FL_REJECT	= 0	n/a	REJECT

Configuration

1. Create a Flexible Parser table using FPP_CMD_FP_TABLE + FPP_ACTION_REGISTER.
2. Create one or multiple rules with FPP_CMD_FP_RULE + FPP_ACTION_REGISTER.
3. Assing rules to tables via FPP_CMD_FP_TABLE + FPP_ACTION_USE_RULE. Rules can be removed from table with FPP_ACTION_UNUSE_RULE.

Created table can be used for instance as argument of Flexible Router. When not needed the table can be deleted with FPP_CMD_FP_TABLE + FPP_ACTION_DEREGISTER and particular rules with FPP_CMD_FP_RULE + FPP_ACTION_DEREGISTER. This cleanup should be always considered since tables and rules are stored in limited PFE internal memory.

Flexible parser classification introduces performance penalty which is proportional to number of rules and complexity of the table.

5.1.7.4 Flexible Router

Introduction

Flexible router specifies behavior when ingress packets are classified and routed according to custom rules different from standard L2 Bridge (Switch) or IPv4/IPv6 Router processing. Feature allows definition of packet distribution rules using physical and logical interfaces. The classification hierarchy is given by ingress physical interface containing a configurable set of logical interfaces. Every time a packet is received via the respective physical interface, which is configured to use the Flexible Router classification, a walk through the list of associated logical interfaces is performed. Every logical interface is used to match the packet using interface-specific rules (fpp_if_m_rules_t). In case of match the matching packet is processed according to the interface configuration (e.g. forwarded via specific physical interface(s), dropped, sent to host, ...). In case when more rules are specified, the logical interface can be configured to apply logical AND or OR to get the match result. Please see the example within Interface Management.

Configuration

1. Lock interface database with FPP_CMD_IF_LOCK_SESSION.
2. Use FPP_CMD_PHY_IF + FPP_ACTION_UPDATE to set a physical interface(s) to FPP_IF_OP_FLEXIBLE_ROUTER operation mode.
3. Use FPP_CMD_LOG_IF + FPP_ACTION_REGISTER to create new logical interface(s) if needed.
4. Optionally, if Flexible Parser is desired to be used as a classification rule, create table(s) according to Flexible Parser description.
5. Configure existing logical interface(s) (set match rules and arguments) via FPP_CMD_LOG_IF + FPP_ACTION_UPDATE.
6. Unlock interface database with FPP_CMD_IF_UNLOCK_SESSION.

Note that Flexible Router can be used to implement certain form of IPv4/IPv6 Router (TCP/UDP) as well as L2 Bridge (Switch). Such usage is of course not recommended since both mentioned features exist as fully optimized implementation and usage of Flexible Router this way would pointlessly affect forwarding performance.

Files

- file fpp_ext.h

  Extension of the legacy fpp.h.
- file libfci.h

  Generic LibFCI header file.

Macros

- #define FPP_CMD_PHY_IF

  FCI command for working with physical interfaces.
- #define FPP_CMD_LOG_IF

  FCI command for working with logical interfaces.
- #define FPP_CMD_IF_LOCK_SESSION

  FCI command to perform lock on interface database.
- #define FPP_CMD_IF_UNLOCK_SESSION

FCI command to perform unlock on interface database.

- #define FPP_CMD_L2_BD

  VLAN-based L2 bridge domain management.
- #define FPP_CMD_FP_TABLE

  Administers the Flexible Parser tables.
- #define FPP_CMD_FP_RULE

  Administers the Flexible Parser rules.
- #define FPP_ACTION_USE_RULE

  Flexible Parser specific 'use' action for FPP_CMD_FP_TABLE.
- #define FPP_ACTION_UNUSE_RULE

  Flexible Parser specific 'unuse' action for FPP_CMD_FP_TABLE.
- #define FPP_CMD_FP_FLEXIBLE_FILTER

  Uses flexible parser to filter out frames from further processing.
- #define FPP_CMD_DATA_BUF_PUT

  FCI command to send an arbitrary data to the accelerator.
- #define FPP_CMD_DATA_BUF_AVAIL

  Event reported when accelerator wants to send a data buffer to host.
- #define FCI_CFG_FORCE_LEGACY_API

  Changes the LibFCI API so it is more compatible with legacy implementation.
- #define FPP_CMD_IPV4_CONNTRACK_CHANGE
- #define FPP_CMD_IPV6_CONNTRACK_CHANGE
- #define CTCMD_FLAGS_ORIG_DISABLED

  Disable connection originator.
- #define CTCMD_FLAGS_REP_DISABLED

  Disable connection replier.

Enumerations

- enum fpp_if_flags_t {
  FPP_IF_ENABLED, FPP_IF_PROMISC,
  FPP_IF_MATCH_OR, FPP_IF_DISCARD,
  FPP_IF_MIRROR }

  Interface flags.
- enum fpp_phy_if_op_mode_t {
  FPP_IF_OP_DISABLED, FPP_IF_OP_DEFAULT,
  FPP_IF_OP_BRIDGE, FPP_IF_OP_ROUTER,
  FPP_IF_OP_VLAN_BRIDGE, FPP_IF_OP_FLEXIBLE_ROUTER }

  Physical if modes.
- enum fpp_if_m_rules_t {
  FPP_IF_MATCH_TYPE_ETH, FPP_IF_MATCH_TYPE_VLAN,
  FPP_IF_MATCH_TYPE_PPPOE, FPP_IF_MATCH_TYPE_ARP,
  FPP_IF_MATCH_TYPE_MCAST, FPP_IF_MATCH_TYPE_IPV4,

FPP_IF_MATCH_TYPE_IPV6, FPP_IF_MATCH_RESERVED7, FPP_IF_MATCH_RESERVED8, FPP_IF_MATCH_TYPE_IPX, FPP_IF_MATCH_TYPE_BCAST, FPP_IF_MATCH_TYPE_UDP, FPP_IF_MATCH_TYPE_TCP, FPP_IF_MATCH_TYPE_ICMP, FPP_IF_MATCH_TYPE_IGMP, FPP_IF_MATCH_VLAN, FPP_IF_MATCH_PROTO, FPP_IF_MATCH_SPORT, FPP_IF_MATCH_DPORT, FPP_IF_MATCH_SIP6, FPP_IF_MATCH_DIP6, FPP_IF_MATCH_SIP, FPP_IF_MATCH_DIP, FPP_IF_MATCH_ETHTYPE, FPP_IF_MATCH_FP0, FPP_IF_MATCH_FP1, FPP_IF_MATCH_SMAC, FPP_IF_MATCH_DMAC, FPP_IF_MATCH_HIF_COOKIE }

Match rules. Can be combined using bitwise OR.

- enum fpp_phy_if_block_state_t { BS_NORMAL, BS_BLOCKED, BS_LEARN_ONLY, BS_FORWARD_ONLY }

  Interface blocking state.
- enum fpp_l2_bd_flags_t { FPP_L2BR_DOMAIN_DEFAULT, FPP_L2BR_DOMAIN_FALLBACK }

  L2 bridge domain flags.
- enum fpp_fp_rule_match_action_t { FP_ACCEPT, FP_REJECT, FP_NEXT_RULE }

  Specifies the Flexible Parser result on the rule match.
- enum fpp_fp_offset_from_t { FP_OFFSET_FROM_L2_HEADER, FP_OFFSET_FROM_L3_HEADER, FP_OFFSET_FROM_L4_HEADER }

  Specifies how to calculate the frame data offset.
- enum fci_mcast_groups_t { FCI_GROUP_NONE, FCI_GROUP_CATCH }

  List of supported multicast groups.
- enum fci_client_type_t { FCI_CLIENT_DEFAULT }

  List of supported FCI client types.
- enum fci_cb_retval_t { FCI_CB_STOP, FCI_CB_CONTINUE }

  The FCI callback return values.

Functions

- FCI_CLIENT * fci_open (fci_client_type_t type, fci_mcast_groups_t group)

  Creates new FCI client and opens a connection to FCI endpoint.
- int fci_close (FCI_CLIENT *client)

  Disconnects from FCI endpoint and destroys FCI client instance.
- int fci_catch (FCI_CLIENT *client)

  Catch and process all FCI messages delivered to the FCI client.

- int fci_cmd (FCI_CLIENT ∗client, unsigned short fcode, unsigned short ∗cmd_buf, unsigned short cmd_len, unsigned short ∗rep_buf, unsigned short ∗rep_len)

  Run an FCI command with optional data response.
- int fci_query (FCI_CLIENT ∗this_client, unsigned short fcode, unsigned short cmd_len, unsigned short ∗pcmd, unsigned short ∗rsplen, unsigned short ∗rsp_data)

  Run an FCI command with data response.
- int fci_write (FCI_CLIENT ∗client, unsigned short fcode, unsigned short cmd_len, unsigned short ∗cmd_buf)

  Run an FCI command.
- int fci_register_cb (FCI_CLIENT ∗client, fci_cb_retval_t(∗event_cb)(unsigned short fcode, unsigned short len, unsigned short ∗payload))

  Register event callback function.
- int fci_fd (FCI_CLIENT ∗this_client)

  Obsolete function, shall not be used.

5.1.8 Defines

5.1.8.1 FPP_CMD_PHY_IF

```
#define FPP_CMD_PHY_IF
```

FCI command for working with physical interfaces.

Note

> Command is defined as extension of the legacy fpp.h.

Interfaces are needed to be known to FCI to support insertion of routes and conntracks. Command can be used to get operation mode, mac address and operation flags (enabled, promisc).

Command can be used with various `.action` values:

- `FPP_ACTION_UPDATE`: Updates properties of an existing physical interface.
- `FPP_ACTION_QUERY`: Gets head of list of existing physical interfaces properties.
- `FPP_ACTION_QUERY_CONT`: Gets next item from list of existing physical interfaces. Shall be called after FPP_ACTION_QUERY was called. On each call it replies with properties of the next interface in the list.

Note

> Precondition to use the query is to atomically lock the access with FPP_CMD_IF_LOCK_SESSION.

Command Argument Type: fpp_phy_if_cmd_t

Action FPP_ACTION_UPDATE

Update interface properties. Set fpp_phy_if_cmd_t.action to FPP_ACTION_UPDATE and fpp_phy_if_cmd_t.name to name of the desired interface to be updated. Rest of the fpp_phy_if_cmd_t members will be considered to be used as the new interface properties. It is recommended to use read-modify-write approach in combination with FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT.

Action FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get interface properties. Set fpp_phy_if_cmd_t.action to FPP_ACTION_QUERY to get first interface from the list of physical interfaces or FPP_ACTION_QUERY_CONT to get subsequent entries. Response data type for query commands is of type fpp_phy_if_cmd_t.

For operation modes see fpp_phy_if_op_mode_t. For operation flags see fpp_if_flags_t.

Possible command return values are:

- `FPP_ERR_OK`: Success.
- `FPP_ERR_IF_ENTRY_NOT_FOUND`: Last entry in the query session.
- `FPP_ERR_IF_WRONG_SESSION_ID`: Someone else is already working with the interfaces.
- `FPP_ERR_INTERNAL_FAILURE`: Internal FCI failure.

Examples

fpp_cmd_phy_if.c.

5.1.8.2 FPP_CMD_LOG_IF

```
#define FPP_CMD_LOG_IF
```

FCI command for working with logical interfaces.

Note

Command is defined as extension of the legacy fpp.h.

Command can be used to update match rules of logical interface or for adding egress interfaces. It can also update operational flags (enabled, promisc, match). Following values of `.action` are supported:

- `FPP_ACTION_REGISTER`: Creates a new logical interface.
- `FPP_ACTION_DEREGISTER`: Destroys an existing logical interface.
- `FPP_ACTION_UPDATE`: Updates properties of an existing logical interface.

- `FPP_ACTION_QUERY`: Gets head of list of existing logical interfaces parameters.
- `FPP_ACTION_QUERY_CONT`: Gets next item from list of existing logical interfaces. Shall be called after FPP_ACTION_QUERY was called. On each call it replies with properties of the next interface.

Precondition to use the query is to atomic lock the access with FPP_CMD_IF_LOCK_SESSION.

Command Argument Type: fpp_log_if_cmd_t

Action FPP_ACTION_REGISTER

To create a new logical interface the FPP_CMD_LOG_IF command expects following values to be set in the command argument structure:

```
fpp_log_if_cmd_t cmd_data =
{
  .action = FPP_ACTION_REGISTER,    // Register new logical interface
  .name = "logif1",                 // Name of the new logical interface
  .parent_name = "emac0"            // Name of the parent physical interface
};
```

The interface *logif1* will be created as child of *emac0* without any configuration and disabled. Names of available physical interfaces can be obtained via FPP_CMD_PHY_IF + FPP_ACTION_QUERY + FPP_ACTION_QUERY_CONT.

Action FPP_ACTION_DEREGISTER

Items to be set in command argument structure to remove a logical interface:

```
fpp_log_if_cmd_t cmd_data =
{
  .action = FPP_ACTION_DEREGISTER,    // Destroy an existing logical interface
  .name = "logif1",                   // Name of the logical interface to destroy
};
```

Action FPP_ACTION_UPDATE

To update logical interface properties just set fpp_log_if_cmd_t.action to FPP_ACTION_UPDATE and fpp_log_if_cmd_t.name to the name of logical interface which you wish to update. Rest of the fpp_log_if_cmd_t structure members will be considered to be used as the new interface properties. It is recommended to use read-modify-write approach in combination with FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT.

For match rules see (fpp_if_m_rules_t). For match rules arguments see (fpp_if_m_args_t).

Possible command return values are:

- `FPP_ERR_OK`: Update successful.
- `FPP_ERR_IF_ENTRY_NOT_FOUND`: If corresponding logical interface doesn't exit.
- `FPP_ERR_IF_RESOURCE_ALREADY_LOCKED`: Someone else is already configuring the interfaces.
- `FPP_ERR_INTERNAL_FAILURE`: Internal FCI failure.

Action FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get interface properties. Set fpp_log_if_cmd_t.action to FPP_ACTION_QUERY to get first interface from the list of all logical interfaces or FPP_ACTION_QUERY_CONT to get subsequent entries. Response data type for query commands is of type fpp_log_if_cmd_t.

Possible command return values are:

- `FPP_ERR_OK`: Success.
- `FPP_ERR_IF_ENTRY_NOT_FOUND`: Last entry in the query session.
- `FPP_ERR_IF_WRONG_SESSION_ID`: Someone else is already working with the interfaces.
- `FPP_ERR_IF_MATCH_UPDATE_FAILED`: Update of match flags has failed.
- `FPP_ERR_IF_EGRESS_UPDATE_FAILED`: Update of egress interfaces has failed.
- `FPP_ERR_IF_EGRESS_DOESNT_EXIST`: Egress interface provided in command doesn't exist.
- `FPP_ERR_IF_OP_FLAGS_UPDATE_FAILED`: Operation flags update has failed (PROMISC/ENABLE/MATCH).
- `FPP_ERR_INTERNAL_FAILURE`: Internal FCI failure.

Examples

fpp_cmd_log_if.c.

5.1.8.3 FPP_CMD_IF_LOCK_SESSION

`#define FPP_CMD_IF_LOCK_SESSION`

FCI command to perform lock on interface database.

The reason for it is guaranteed atomic operation between fci/rpc/platform.

Note

Command is defined as extension of the legacy fpp.h.

Possible command return values are:

- FPP_ERR_OK: Lock successful
- FPP_ERR_IF_RESOURCE_ALREADY_LOCKED: Database was already locked by someone else

Examples

fpp_cmd_log_if.c, and fpp_cmd_phy_if.c.

5.1.8.4 FPP_CMD_IF_UNLOCK_SESSION

```
#define FPP_CMD_IF_UNLOCK_SESSION
```

FCI command to perform unlock on interface database.

The reason for it is guaranteed atomic operation between fci/rpc/platform.

Note

Command is defined as extension of the legacy fpp.h.

Possible command return values are:

- FPP_ERR_OK: Lock successful
- FPP_ERR_IF_WRONG_SESSION_ID: The lock wasn't locked or was locked in different session and will not be unlocked.

Examples

fpp_cmd_log_if.c, and fpp_cmd_phy_if.c.

5.1.8.5 FPP_CMD_L2_BD

```
#define FPP_CMD_L2_BD
```

VLAN-based L2 bridge domain management.

Bridge domain can be used to include a set of physical interfaces and isolate them from another domains using VLAN. Command can be used with various `.action` values:

- `FPP_ACTION_REGISTER`: Create a new bridge domain.
- `FPP_ACTION_DEREGISTER`: Delete bridge domain.
- `FPP_ACTION_UPDATE`: Update a bridge domain meaning that will rewrite domain properties except of VLAN ID.
- `FPP_ACTION_QUERY`: Gets head of list of registered domains.
- `FPP_ACTION_QUERY_CONT`: Get next item from list of registered domains. Shall be called after `FPP_ACTION_QUERY` was called. On each call it replies with parameters of next domain. It returns `FPP_ERR_RT_ENTRY_NOT_FOUND` when no more entries exist.

Command Argument Type: fpp_l2_bd_cmd_t

Action FPP_ACTION_REGISTER

Items to be set in command argument structure:

```
fpp_l2_bd_cmd_t cmd_data =
{
  // Register new bridge domain
  .action = FPP_ACTION_REGISTER,
  // VLAN ID associated with the domain (network endian)
  .vlan = ...,
  // Action to be taken when destination MAC address (uni-cast) of a packet
  // matching the domain is found in the MAC table: 0 - Forward, 1 - Flood,
  // 2 - Punt, 3 - Discard
  .ucast_hit = ...,
  // Action to be taken when destination MAC address (uni-cast) of a packet
  // matching the domain is not found in the MAC table.
  .ucast_miss = ...,
  // Multicast hit action
  .mcast_hit = ...,
  // Multicast miss action
  .mcast_miss = ...
};
```

Possible command return values are:

- `FPP_ERR_OK`: Domain added.
- `FPP_ERR_WRONG_COMMAND_PARAM`: Unexpected argument.
- `FPP_ERR_L2BRIDGE_DOMAIN_ALREADY_REGISTERED`: Given domain already registered.
- `FPP_ERR_INTERNAL_FAILURE`: Internal FCI failure.

Action FPP_ACTION_DEREGISTER

Items to be set in command argument structure:

```
fpp_l2_bd_cmd_t cmd_data =
{
  // Delete bridge domain
  .action = FPP_ACTION_DEREGISTER,
  // VLAN ID associated with the domain to be deleted (network endian)
  .vlan = ...,
};
```

Possible command return values are:

- `FPP_ERR_OK`: Domain removed.
- `FPP_ERR_WRONG_COMMAND_PARAM`: Unexpected argument.
- `FPP_ERR_L2BRIDGE_DOMAIN_NOT_FOUND`: Given domain not found.
- `FPP_ERR_INTERNAL_FAILURE`: Internal FCI failure.

Action FPP_ACTION_UPDATE

Items to be set in command argument structure:

```
fpp_l2_bd_cmd_t cmd_data =
{
  // Update bridge domain
  .action = FPP_ACTION_UPDATE,
```

```
  // VLAN ID associated with the domain to be updated (network endian)
  .vlan = ...,
  // New unicast hit action (0 - Forward, 1 - Flood, 2 - Punt, 3 - Discard)
  .ucast_hit = ...,
  // New unicast miss action
  .ucast_miss = ...,
  // New multicast hit action
  .mcast_hit = ...,
  // New multicast miss action
  .mcast_miss = ...,
  // New port list (network endian). Bitmask where every set bit represents
  // ID of physical interface being member of the domain. For instance bit
  // (1 « 3), if set, says that interface with ID=3 is member of the domain.
  // Only valid interface IDs are accepted by the command. If flag is set,
  // interface is added to the domain. If flag is not set and interface
  // has been previously added, it is removed. The IDs are given by the
  // related FCI endpoint and related networking HW. Interface IDs can be
  // obtained via FPP_CMD_PHY_IF.
  .if_list = ...,
  // Flags marking interfaces listed in @c if_list to be 'tagged' or
  // 'untagged' (network endian). If respective flag is set, corresponding
  // interface within the @c if_list is treated as 'untagged' meaning that
  // the VLAN tag will be removed. Otherwise it is configured as 'tagged'.
  // Note that only interfaces listed within the @c if_list are taken into
  // account.
  .untag_if_list = ...,
};
```

Possible command return values are:

- `FPP_ERR_OK`: Domain updated.
- `FPP_ERR_WRONG_COMMAND_PARAM`: Unexpected argument.
- `FPP_ERR_L2BRIDGE_DOMAIN_NOT_FOUND`: Given domain not found.
- `FPP_ERR_INTERNAL_FAILURE`: Internal FCI failure.

Action FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Items to be set in command argument structure:

```
fpp_l2_bd_cmd_t cmd_data =
{
  .action = ...     // Either FPP_ACTION_QUERY or FPP_ACTION_QUERY_CONT
};
```

Response data type for queries: fpp_l2_bd_cmd_t

Response data provided (all values in network byte order):

```
// VLAN ID associated with domain (network endian)
rsp_data.vlan;
// Action to be taken when destination MAC address (uni-cast) of a packet
// matching the domain is found in the MAC table: 0 - Forward, 1 - Flood,
// 2 - Punt, 3 - Discard
rsp_data.ucast_hit;
// Action to be taken when destination MAC address (uni-cast) of a packet
// matching the domain is not found in the MAC table.
rsp_data.ucast_miss;
// Multicast hit action.
rsp_data.mcast_hit;
// Multicast miss action.
rsp_data.mcast_miss;
// Bitmask where every set bit represents ID of physical interface being member
// of the domain. For instance bit (1 « 3), if set, says that interface with ID=3
// is member of the domain.
rsp_data.if_list;
// Similar to @c if_list but this interfaces are configured to be VLAN 'untagged'.
rsp_data.untag_if_list;
```

```
// See the fpp_l2_bd_flags_t.
rsp_data.flags;
```

Possible command return values are:

- FPP_ERR_OK: Response buffer written.
- FPP_ERR_L2BRIDGE_DOMAIN_NOT_FOUND: No more entries.
- FPP_ERR_INTERNAL_FAILURE: Internal FCI failure.

5.1.8.6 FPP_CMD_FP_TABLE

```
#define FPP_CMD_FP_TABLE
```

Administers the Flexible Parser tables.

The Flexible Parser table is an ordered set of Flexible Parser rules which are matched in the order of appearance until match occurs or end of the table is reached. The following actions can be done on the table:

- FPP_ACTION_REGISTER: Create a new table with a given name.
- FPP_ACTION_DEREGISTER: Destroy an existing table.
- FPP_ACTION_USE_RULE: Add a rule into the table at specified position.
- FPP_ACTION_UNUSE_RULE: Remove a rule from the table.
- FPP_ACTION_QUERY: Return the first rule in the table.
- FPP_ACTION_QUERY_CONT: Return the next rule in the table.

The Flexible Parser starts processing the table from the 1st rule in the table. If there is no match the Flexible Parser always continues with the rule following the currently processed rule. The processing ends once rule match happens and the rule action is one of the FP_ACCEPT or FP_REJECT and the respective value is returned. REJECT is also returned after the last rule in the table was processed without any match. The Flexible Parser may branch to arbitrary rule in the table if some rule matches and the action is FP_NEXT_RULE. Note that loops are forbidden.

See the FPP_CMD_FP_RULE and fpp_fp_rule_props_t for the detailed description of how the rules are being matched.

Action FPP_ACTION_REGISTER

Items to be set in command argument structure:

```
fpp_fp_table_cmd_t cmd_data =
{
    .action = FPP_ACTION_REGISTER,       // Add a new table
    .t.table_name = "table_name",       // Unique up-to-15-character table identifier
};
```

Action FPP_ACTION_DEREGISTER

Items to be set in command argument structure:
```
fpp_fp_table_cmd_t cmd_data =
{
   .action = FPP_ACTION_DEREGISTER,  // Remove an existing table
   .t.table_name = "table_name",     // Identifier of the table to be destroyed
};
```

Action FPP_ACTION_USE_RULE

Items to be set in command argument structure:
```
fpp_fp_table_cmd_t cmd_data =
{
   .action = FPP_ACTION_USE_RULE,    // Add existing rule into specified table
   .t.table_name = "table_name",     // Identifier of the table to add the rule
   .t.rule_name = "rule_name",       // Identifier of the rule to be added into the table
};
```

Note

Single rule can be member of only one table.

Action FPP_ACTION_UNUSE_RULE

Items to be set in command argument structure:
```
fpp_flexible_parser_table_cmd cmd_data =
{
   .action = FPP_ACTION_UNUSE_RULE,  // Remove an existing table
   .t.rule_name = "rule_name",       // Identifier of the rule to be removed from the table
};
```

Action FPP_ACTION_QUERY

Items to be set in command argument structure:
```
fpp_flexible_parser_table_cmd cmd_data =
{
   .action = FPP_ACTION_QUERY,       // Start query of the table rules
   .t.table_name = "table_name",     // Identifier of the table to be queried
};
```

Response data type for queries: fpp_fp_rule_cmd_t

Response data provided:
```
rsp_data.r.name;             // Name of the rule
rsp_data.r.data;             // Expected data value (network endian)
rsp_data.r.mask;             // Mask to be applied on frame data (network endian)
rsp_data.r.offset;           // Offset of the data in the frame (network endian)
rsp_data.r.invert;           // Invert match or not
rsp_data.r.match_action;     // Action to be done on match
rsp_data.r.next_rule_name;   // Next rule to be examined if match_action == FP_NEXT_RULE
```

Note

All data is provided in the network byte order.

Action FPP_ACTION_QUERY_CONT

Items to be set in command argument structure:

```
fpp_flexible_parser_table_cmd cmd_data =
{
   .action = FPP_ACTION_QUERY_CONT,     // Continue query of the table rules by the next rule
   .t.table_name = "table_name",        // Identifier of the table to be queried
};
```

Response data is provided in the same form as for FPP_ACTION_QUERY action.

5.1.8.7 FPP_CMD_FP_RULE

```
#define FPP_CMD_FP_RULE
```

Administers the Flexible Parser rules.

Each Flexible Parser rule consists of a condition specified by `data`, `mask` and `offset` triplet and action to be performed. If 32-bit frame data at given `offset` masked by `mask` is equal to the specified `data` masked by the same `mask` then the condition is true. An invert flag may be set to invert the condition result. The rule action may be either `accept`, `reject` or `next_rule` which means to continue with a specified rule.

The rule administering command may be one of the following actions:

- `FPP_ACTION_REGISTER`: Create a new rule.
- `FPP_ACTION_DEREGISTER`: Delete an existing rule.
- `FPP_ACTION_QUERY`: Return the first rule (among all existing rules).
- `FPP_ACTION_QUERY_CONT`: Return the next rule.

Action FPP_ACTION_REGISTER

Items to be set in command argument structure:

```
fpp_fp_rule_cmd_t cmd_data =
{
   // Creates a new rule
   .action = FPP_ACTION_REGISTER,
   // Unique up-to-15-character rule identifier
   .r.rule_name = "rule_name",
   // 32-bit data to match with the frame data at given offset (network endian)
   .r.data = htonl(0x08000000),
   // 32-bit mask to apply on the frame data and .r.data before comparison (network endian)
   .r.mask = htonl(0xFFFF0000),
   // Offset of the frame data to be compared (network endian)
   .r.offset = htonl(12),
   // Invert match or not (values 0 or 1)
   .r.invert = 0,
   // How to calculate the offset
   .r.match_action = FP_OFFSET_FROM_L2_HEADER,
   // Action to be done on match
   .r.offset_from = FP_ACCEPT,
   // Identifier of the next rule to use when match_action == FP_NEXT_RULE
   .r.next_rule_name = "rule_name2"
};
```

This example is used to match and accept all IPv4 frames (16-bit value 0x0800 at bytes 12 and 13, when starting bytes counting from 0).

Note

> All values are specified in the network byte order.

Warning

It is forbidden to create rule loops using the *next_rule* feature.

Action FPP_ACTION_DEREGISTER

Items to be set in command argument structure:
```
fpp_fp_rule_cmd_t cmd_data =
{
    .action = FPP_ACTION_DEREGISTER,    // Deletes an existing rule
    .r.rule_name = "rule_name",         // Identifier of the rule to be deleted
};
```

Action FPP_ACTION_QUERY

Items to be set in command argument structure:
```
fpp_flexible_parser_rule_cmd cmd_data =
{
    .action = FPP_ACTION_QUERY        // Start the rules query
};
```
Response data type for queries: fpp_fp_rule_cmd_t

Response data provided:
```
rsp_data.r.name;                // Name of the rule
rsp_data.r.data;                // Expected data value (network endian)
rsp_data.r.mask;                // Mask to be applied on frame data (network endian)
rsp_data.r.offset;              // Offset of the data in the frame (network endian)
rsp_data.r.invert;              // Invert match or not
rsp_data.r.match_action;        // Action to be done on match
rsp_data.r.next_rule_name;      // Next rule to be examined if match_action == FP_NEXT_RULE
```

Note

All data is provided in the network byte order.

Action FPP_ACTION_QUERY_CONT

Items to be set in command argument structure:
```
fpp_flexible_parser_rule_cmd cmd_data =
{
    .action = FPP_ACTION_QUERY_CONT    // Continue with the rules query
};
```
Response data is provided in the same form as for FPP_ACTION_QUERY action.

5.1.8.8 FPP_CMD_FP_FLEXIBLE_FILTER

```
#define FPP_CMD_FP_FLEXIBLE_FILTER
```

Uses flexible parser to filter out frames from further processing.

Allows registration of a Flexible Parser table (see FPP_CMD_FP_TABLE) as a filter which:

- `FPP_ACTION_REGISTER`: Use the specified table as a Flexible Filter (replace old table by a new one if already configured).
- `FPP_ACTION_DEREGISTER`: Disable Flexible Filter, no table will be used as Flexible Filter.

The Flexible Filter examines received frames before any other processing and discards those which have REJECT result from the configured Flexible Parser.

See the FPP_CMD_FP_TABLE for flexible parser behavior description.

Action FPP_ACTION_REGISTER

Items to be set in command argument structure:
```
fpp_flexible_filter_cmd_t cmd_data =
{
   // Set the specified table as Flexible Filter
   .action = FPP_ACTION_REGISTER,
   // Name of the Flexible Parser table to be used to filter the frames
   .table_name = "table_name"
}
```

Action FPP_ACTION_DEREGISTER

Items to be set in command argument structure:
```
fpp_flexible_filter_cmd_t cmd_data =
{
   // Disable the Flexible Filter
   .action = FPP_ACTION_DEREGISTER,
}
```

5.1.8.9 FPP_CMD_DATA_BUF_PUT

```
#define FPP_CMD_DATA_BUF_PUT
```

FCI command to send an arbitrary data to the accelerator.

Command is intended to be used to send custom data to the accelerator. Format of the command argument is given by the fpp_buf_cmd_t structure which also defines the maximum payload length. Subsequent commands are not successful until the accelerator reads and acknowledges the current request.

Items to be set in command argument structure:
```
fpp_buf_cmd_t cmd_data =
{
  // Specify buffer payload
  .payload = ...,
  // Payload length in number of bytes
  .len = ...,
};
```

Possible command return values are:

- `FPP_ERR_OK`: Data written and available to the accelerator
- `FPP_ERR_AGAIN`: Previous command has not been finished yet
- `FPP_ERR_INTERNAL_FAILURE`: Internal FCI failure

5.1.8.10 FPP_CMD_DATA_BUF_AVAIL

```
#define FPP_CMD_DATA_BUF_AVAIL
```

Event reported when accelerator wants to send a data buffer to host.

Indication of this event also carries the buffer payload and payload length. Both are available via the event callback arguments (see the callback type and arguments within description of fci_register_cb()).

5.1.8.11 FCI_CFG_FORCE_LEGACY_API

```
#define FCI_CFG_FORCE_LEGACY_API
```

Changes the LibFCI API so it is more compatible with legacy implementation.

LibFCI API was modified to avoid some inconvenient properties. Here are the points the legacy API differs in:

1. With legacy API, argument `rsp_data` of function `fci_query` shall be provided shifted by two bytes this way:
   ```
   reply_struct_t rsp_data;
   retval = fci_query(this_client, fcode, cmd_len, &pcmd, &rsplen, (unsigned short
    *)(&rsp_data) + 1u);
   ```
   Where `reply_struct_t` is the structure type depending on command being called.

2. In legacy API, macros FPP_CMD_IPV4_CONNTRACK_CHANGE and FPP_CMD_IPV6_CONNTRACK_CHANGE are defined in application files. In current API they are defined here in libfci.h.

Warning

It is not recommended to enable this feature.

5.1.8.12 FPP_CMD_IPV4_CONNTRACK_CHANGE

```
#define FPP_CMD_IPV4_CONNTRACK_CHANGE
```

5.1.8.13 FPP_CMD_IPV6_CONNTRACK_CHANGE

```
#define FPP_CMD_IPV6_CONNTRACK_CHANGE
```

5.1.8.14 CTCMD_FLAGS_REP_DISABLED

```
#define CTCMD_FLAGS_REP_DISABLED
```

Disable connection replier.

Used to create uni-directional connections (see FPP_CMD_IPV4_CONNTRACK, FPP_CMD_IPV4_CONNTRACK)

Examples

fpp_cmd_ipv4_conntrack.c, and fpp_cmd_ipv6_conntrack.c.

5.1.9 Enums

5.1.9.1 fpp_if_flags_t

```
enum fpp_if_flags_t
```

Interface flags.

Enumerator

FPP_IF_ENABLED	If set, interface is enabled
FPP_IF_PROMISC	If set, interface is promiscuous
FPP_IF_MATCH_OR	Result of match is logical OR of rules, else AND
FPP_IF_DISCARD	Discard matching frames
FPP_IF_MIRROR	If set mirroring is enabled

5.1.9.2 fpp_phy_if_op_mode_t

```
enum fpp_phy_if_op_mode_t
```

Physical if modes.

Enumerator

FPP_IF_OP_DISABLED	Disabled
FPP_IF_OP_DEFAULT	Default operational mode
FPP_IF_OP_BRIDGE	L2 bridge
FPP_IF_OP_ROUTER	L3 router
FPP_IF_OP_VLAN_BRIDGE	L2 bridge with VLAN
FPP_IF_OP_FLEXIBLE_ROUTER	Flexible router

5.1.9.3 fpp_if_m_rules_t

`enum fpp_if_m_rules_t`

Match rules. Can be combined using bitwise OR.

Enumerator

FPP_IF_MATCH_TYPE_ETH	Match ETH Packets
FPP_IF_MATCH_TYPE_VLAN	Match VLAN Tagged Packets
FPP_IF_MATCH_TYPE_PPPOE	Match PPPoE Packets
FPP_IF_MATCH_TYPE_ARP	Match ARP Packets
FPP_IF_MATCH_TYPE_MCAST	Match Multicast (L2) Packets
FPP_IF_MATCH_TYPE_IPV4	Match IPv4 Packets
FPP_IF_MATCH_TYPE_IPV6	Match IPv6 Packets
FPP_IF_MATCH_RESERVED7	Reserved
FPP_IF_MATCH_RESERVED8	Reserved
FPP_IF_MATCH_TYPE_IPX	Match IPX Packets
FPP_IF_MATCH_TYPE_BCAST	Match Broadcast (L2) Packets
FPP_IF_MATCH_TYPE_UDP	Match UDP Packets
FPP_IF_MATCH_TYPE_TCP	Match TCP Packets
FPP_IF_MATCH_TYPE_ICMP	Match ICMP Packets
FPP_IF_MATCH_TYPE_IGMP	Match IGMP Packets
FPP_IF_MATCH_VLAN	Match VLAN ID
FPP_IF_MATCH_PROTO	Match IP Protocol
FPP_IF_MATCH_SPORT	Match L4 Source Port
FPP_IF_MATCH_DPORT	Match L4 Destination Port
FPP_IF_MATCH_SIP6	Match Source IPv6 Address
FPP_IF_MATCH_DIP6	Match Destination IPv6 Address
FPP_IF_MATCH_SIP	Match Source IPv4 Address
FPP_IF_MATCH_DIP	Match Destination IPv4 Address
FPP_IF_MATCH_ETHTYPE	Match EtherType
FPP_IF_MATCH_FP0	Match Packets Accepted by Flexible Parser 0
FPP_IF_MATCH_FP1	Match Packets Accepted by Flexible Parser 1
FPP_IF_MATCH_SMAC	Match Source MAC Address
FPP_IF_MATCH_DMAC	Match Destination MAC Address
FPP_IF_MATCH_HIF_COOKIE	Match HIF header cookie value

5.1.9.4 fpp_phy_if_block_state_t

`enum fpp_phy_if_block_state_t`

Interface blocking state.

Enumerator

BS_NORMAL	Learning and forwarding enabled
BS_BLOCKED	Learning and forwarding disabled
BS_LEARN_ONLY	Learning enabled, forwarding disabled
BS_FORWARD_ONLY	Learning disabled, forwarding enabled

5.1.9.5 fpp_l2_bd_flags_t

`enum fpp_l2_bd_flags_t`

L2 bridge domain flags.

Enumerator

FPP_L2BR_DOMAIN_DEFAULT	Domain type is default
FPP_L2BR_DOMAIN_FALLBACK	Domain type is fallback

5.1.9.6 fpp_fp_rule_match_action_t

`enum fpp_fp_rule_match_action_t`

Specifies the Flexible Parser result on the rule match.

Enumerator

FP_ACCEPT	Flexible parser result on rule match is ACCEPT
FP_REJECT	Flexible parser result on rule match is REJECT
FP_NEXT_RULE	On rule match continue matching by the specified rule

5.1.9.7 fpp_fp_offset_from_t

`enum fpp_fp_offset_from_t`

Specifies how to calculate the frame data offset.

The offset may be calculated either from the L2, L3 or L4 header beginning. The L2 header beginning is also the Ethernet frame beginning because the Ethernet frame begins with the L2 header. This offset is always valid however if the L3 or L4 header is not recognized then the rule is always skipped as not-matching.

Enumerator

FP_OFFSET_FROM_L2_HEADER	Calculate offset from the L2 header (frame beginning)
FP_OFFSET_FROM_L3_HEADER	Calculate offset from the L3 header
FP_OFFSET_FROM_L4_HEADER	Calculate offset from the L4 header

5.1.9.8 fci_mcast_groups_t

`enum fci_mcast_groups_t`

List of supported multicast groups.

An FCI client instance can be member of a multicast group. It means it can send and receive multicast messages to/from another group members (another FCI instances or FCI endpoints). This can be in most cases used by FCI endpoint to notify all associated FCI instances about some event has occurred.

Note

> Each group is intended to be represented by a single bit flag (max 32-bit, so it is possible to have max 32 multicast groups). Then, groups can be combined using bitwise OR operation.

Enumerator

FCI_GROUP_NONE	Default MCAST group value, no group, for sending FCI commands
FCI_GROUP_CATCH	MCAST group for catching events

5.1.9.9 fci_client_type_t

`enum fci_client_type_t`

List of supported FCI client types.

FCI client can specify using this type to which FCI endpoint shall be connected.

Enumerator

FCI_CLIENT_DEFAULT	Default type (equivalent of legacy FCILIB_FF_TYPE macro)

5.1.9.10 fci_cb_retval_t

`enum fci_cb_retval_t`

The FCI callback return values.

These return values shall be used in FCI callback (see fci_register_cb). It tells fci_catch function whether it should return or continue.

Enumerator

FCI_CB_STOP	Stop waiting for events and exit fci_catch function
FCI_CB_CONTINUE	Continue waiting for next events

5.1.10 Functions

5.1.10.1 fci_open()

```
FCI_CLIENT* fci_open (
        fci_client_type_t type,
        fci_mcast_groups_t group )
```

Creates new FCI client and opens a connection to FCI endpoint.

Binds the FCI client with FCI endpoint. This enables sending/receiving data to/from the endpoint. Refer to the remaining API for possible communication options.

Parameters

in	*type*	Client type. Default value is FCI_CLIENT_DEFAULT. See fci_client_type_t.
in	*group*	A 32-bit multicast group mask. Each bit represents single multicast address. FCI instance will listen to specified multicast addresses as well it will send data to all specified multicast groups. See fci_mcast_groups_t.

Returns

The FCI client instance or NULL if failed

5.1.10.2 fci_close()

```
int fci_close (
        FCI_CLIENT * client )
```

Disconnects from FCI endpoint and destroys FCI client instance.

Terminate the FCI client and release all allocated resources.

Parameters

in	*client*	The FCI client instance

Returns

0 if success, error code otherwise

5.1.10.3 fci_catch()

```
int fci_catch (
          FCI_CLIENT * client )
```

Catch and process all FCI messages delivered to the FCI client.

Function is intended to be called in its own thread. It waits for message/event reception. If there is an event callback associated with the FCI client, assigned by function fci_register_cb(), then, when message is received, the callback is called to process the data. As long as there is no error and the callback returns FCI_CB_CONTINUE, fci_catch() continues waiting for another message. Otherwise it returns.

Note

This is a blocking function.

Multicast group FCI_GROUP_CATCH shall be used when opening the client for catching messages

See also

fci_register_cb()

Parameters

in	*client*	The FCI client instance

Returns

0 if success, error code otherwise

5.1.10.4 fci_cmd()

```
int fci_cmd (
          FCI_CLIENT * client,
          unsigned short fcode,
          unsigned short * cmd_buf,
          unsigned short cmd_len,
          unsigned short * rep_buf,
          unsigned short * rep_len )
```

Run an FCI command with optional data response.

This routine can be used when one need to perform any command either with or without data response. If the command responded with some data structure the structure is written into the rep_buf. The length of the returned data structure (number of bytes) is written into rep_len.

Note

The `rep_buf` buffer must be aligned to 4.

Parameters

`in`	*client*	The FCI client instance
`in`	*fcode*	Command to be executed. Available commands are listed in Commands Summary.
`in`	*cmd_buf*	Pointer to structure holding command arguments.
`in`	*cmd_len*	Length of the command arguments structure in bytes.
`out`	*rep_buf*	Pointer to memory where the data response shall be written. Can be NULL.
`in,out`	*rep_len*	Pointer to variable where number of response bytes shall be written.

Return values

<0	Failed to execute the command.
>=0	Command was executed with given return value (`FPP_ERR_OK` for success).

5.1.10.5 fci_query()

```
int fci_query (
          FCI_CLIENT * this_client,
          unsigned short fcode,
          unsigned short cmd_len,
          unsigned short * pcmd,
          unsigned short * rsplen,
          unsigned short * rsp_data )
```

Run an FCI command with data response.

This routine can be used when one need to perform a command which is resulting in a data response. It is suitable for various 'query' commands like reading of whole tables or structured entries from the endpoint.

Note

If either `rsp_data` or `rsplen` is NULL pointer, the response data is discarded.

Parameters

in	*this_client*	The FCI client instance
in	*fcode*	Command to be executed. Available commands are listed in Commands Summary.
in	*cmd_len*	Length of the command arguments structure in bytes
in	*pcmd*	Pointer to structure holding command arguments.
out	*rsplen*	Pointer to memory where length of the data response will be provided
out	*rsp_data*	Pointer to memory where the data response shall be written.

Return values

<0	Failed to execute the command.
>=0	Command was executed with given return value (`FPP_ERR_OK` for success).

Examples

fpp_cmd_ip_route.c, fpp_cmd_log_if.c, and fpp_cmd_phy_if.c.

5.1.10.6 fci_write()

```
int fci_write (
            FCI_CLIENT * client,
            unsigned short fcode,
            unsigned short cmd_len,
            unsigned short * cmd_buf )
```

Run an FCI command.

Similar as the fci_query() but without data response. The endpoint receiving the command is still responsible for generating response but the response is not delivered to the caller.

Parameters

in	*client*	The FCI client instance
in	*fcode*	Command to be executed. Available commands are listed in Commands Summary.
in	*cmd_len*	Length of the command arguments structure in bytes
in	*cmd_buf*	Pointer to structure holding command arguments

Return values

<0	Failed to execute the command.
>=0	Command was executed with given return value (`FPP_ERR_OK` for success).

Examples

fpp_cmd_ip_route.c, fpp_cmd_ipv4_conntrack.c, fpp_cmd_ipv6_conntrack.c, fpp_cmd_log_if.c, and fpp_cmd_phy_if.c.

5.1.10.7 fci_register_cb()

```
int fci_register_cb (
          FCI_CLIENT * client,
          fci_cb_retval_t(*)(unsigned short fcode, unsigned short len,
unsigned short *payload) event_cb )
```

Register event callback function.

FCI endpoint can send various asynchronous messages to the FCI client. In such case, a callback registered via this function is executed if fci_catch() is running.

Parameters

in	*client*	The FCI client instance
in	*event_cb*	The callback function to be executed. When called then `fcode` specifies event code (available events are listed in Events summary), `payload` is pointer to event payload and the `len` is number of bytes in the payload buffer.

Returns

0 if success, error code otherwise

Note

In order to continue receiving messages, the callback function shall always return FCI_CB_CONTINUE. Any other value will cause the fci_catch to return.

Chapter 6

Data Structure Documentation

6.1 FCI_CLIENT Struct Reference

The FCI client representation type.

```
#include <libfci.h>
```

6.1.1 Detailed Description

The FCI client representation type.

This is the FCI instance representation. It is used by the rest of the API to communicate with associated endpoint. The endpoint can be a standalone application/driver taking care of HW configuration tasks and shall be able to interpret commands sent via the LibFCI API.

Examples

fpp_cmd_ip_route.c, fpp_cmd_ipv4_conntrack.c, fpp_cmd_ipv6_conntrack.c, fpp_cmd_log_if.c, and fpp_cmd_phy_if.c.

The documentation for this struct was generated from the following file:

- libfci.h

6.2 fpp_algo_stats_t Struct Reference

Algorithm statistics.

```
#include <fpp_ext.h>
```

Data Fields

- uint32_t processed
- uint32_t accepted
- uint32_t rejected
- uint32_t discarded

6.2.1 Detailed Description

Algorithm statistics.

Statistics used by algorithms in class (eg. log ifs).

Note

All statistics counters are in network byte order.

6.2.2 Field Documentation

6.2.2.1 processed

`uint32_t processed`

Number of frames processed regardless the result

6.2.2.2 accepted

`uint32_t accepted`

Number of frames matching the selection criteria

6.2.2.3 rejected

`uint32_t rejected`

Number of frames not matching the selection criteria

6.2.2.4 discarded

`uint32_t discarded`

Number of frames marked to be dropped

The documentation for this struct was generated from the following file:

- fpp_ext.h

6.3 fpp_buf_cmd_t Struct Reference

Argument structure for the FPP_CMD_DATA_BUF_PUT command.

`#include <fpp_ext.h>`

Data Fields

- uint8_t payload [64]
- uint8_t len

6.3.1 Detailed Description

Argument structure for the FPP_CMD_DATA_BUF_PUT command.

6.3.2 Field Documentation

6.3.2.1 payload

```
uint8_t payload[64]
```

The payload area

6.3.2.2 len

```
uint8_t len
```

Payload length in number of bytes

The documentation for this struct was generated from the following file:

- fpp_ext.h

6.4 fpp_ct6_cmd_t Struct Reference

Data structure used in various functions for IPv6 conntrack management.

```
#include <fpp.h>
```

Data Fields

- uint16_t action
- uint32_t saddr [4]
- uint32_t daddr [4]
- uint16_t sport
- uint16_t dport
- uint32_t saddr_reply [4]
- uint32_t daddr_reply [4]
- uint16_t sport_reply
- uint16_t dport_reply

- uint16_t protocol
- uint16_t flags
- uint32_t route_id
- uint32_t route_id_reply

6.4.1 Detailed Description

Data structure used in various functions for IPv6 conntrack management.

It can be used:

- for command buffer in functions fci_write, fci_query or fci_cmd, with FPP_CMD_IPV6_CONNTRACK command.

Examples

fpp_cmd_ipv6_conntrack.c.

6.4.2 Field Documentation

6.4.2.1 action

`uint16_t action`

Action to perform

Examples

fpp_cmd_ipv6_conntrack.c.

6.4.2.2 saddr

`uint32_t saddr[4]`

Source IP address

6.4.2.3 daddr

`uint32_t daddr[4]`

Destination IP address

6.4.2.4 sport

`uint16_t sport`

Source port

6.4.2.5 dport

`uint16_t dport`

Destination port

6.4.2.6 saddr_reply

`uint32_t saddr_reply[4]`

Source IP address in 'reply' direction

6.4.2.7 daddr_reply

`uint32_t daddr_reply[4]`

Destination IP address in 'reply' direction

6.4.2.8 sport_reply

`uint16_t sport_reply`

Source port in 'reply' direction

6.4.2.9 dport_reply

`uint16_t dport_reply`

Destination port in 'reply' direction

6.4.2.10 protocol

`uint16_t protocol`

Protocol ID: TCP, UDP

6.4.2.11 flags

`uint16_t flags`

Flags. See FPP_CMD_IPV6_CONNTRACK.

6.4.2.12 route_id

`uint32_t route_id`

Associated route ID. See FPP_CMD_IP_ROUTE.

6.4.2.13 route_id_reply

`uint32_t route_id_reply`

Route for 'reply' direction. Applicable only for bi-directional connections.

The documentation for this struct was generated from the following file:

- fpp.h

6.5 fpp_ct_cmd_t Struct Reference

Data structure used in various functions for conntrack management.

`#include <fpp.h>`

Data Fields

- uint16_t action
- uint32_t saddr
- uint32_t daddr
- uint16_t sport
- uint16_t dport
- uint32_t saddr_reply
- uint32_t daddr_reply
- uint16_t sport_reply
- uint16_t dport_reply
- uint16_t protocol
- uint16_t flags
- uint32_t route_id
- uint32_t route_id_reply

6.5.1 Detailed Description

Data structure used in various functions for conntrack management.

It can be used:

- for command buffer in functions fci_write, fci_query or fci_cmd, with FPP_CMD_IPV4_CONNTRACK command.

Examples

fpp_cmd_ipv4_conntrack.c.

6.5.2 Field Documentation

6.5.2.1 action

`uint16_t action`

Action to perform

Examples

fpp_cmd_ipv4_conntrack.c.

6.5.2.2 saddr

`uint32_t saddr`

Source IP address

6.5.2.3 daddr

`uint32_t daddr`

Destination IP address

6.5.2.4 sport

`uint16_t sport`

Source port

6.5.2.5 dport

`uint16_t dport`

Destination port

6.5.2.6 saddr_reply

`uint32_t saddr_reply`

Source IP address in 'reply' direction

6.5.2.7 daddr_reply

`uint32_t daddr_reply`

Destination IP address in 'reply' direction

6.5.2.8 sport_reply

`uint16_t sport_reply`

Source port in 'reply' direction

6.5.2.9 dport_reply

`uint16_t dport_reply`

Destination port in 'reply' direction

6.5.2.10 protocol

`uint16_t protocol`

Protocol ID: TCP, UDP

6.5.2.11 flags

`uint16_t flags`

Flags. See FPP_CMD_IPV4_CONNTRACK.

6.5.2.12 route_id

`uint32_t route_id`

Associated route ID. See FPP_CMD_IP_ROUTE.

6.5.2.13 route_id_reply

`uint32_t route_id_reply`

Route for 'reply' direction. Applicable only for bi-directional connections.

The documentation for this struct was generated from the following file:

- fpp.h

6.6 fpp_fp_rule_cmd_t Struct Reference

Arguments for the FPP_CMD_FP_RULE command.

`#include <fpp_ext.h>`

Data Fields

- uint16_t action
- fpp_fp_rule_props_t r

6.6.1 Detailed Description

Arguments for the FPP_CMD_FP_RULE command.

6.6.2 Field Documentation

6.6.2.1 action

```
uint16_t action
```

Action to be done

6.6.2.2 r

```
fpp_fp_rule_props_t r
```

Parameters of the rule

The documentation for this struct was generated from the following file:

- fpp_ext.h

6.7 fpp_fp_rule_props_t Struct Reference

Properties of the Flexible parser rule.

```
#include <fpp_ext.h>
```

6.7.1 Detailed Description

Properties of the Flexible parser rule.

The rule match can be described as:

```
((frame_data[offset] & mask) == (data & mask)) ? match = true : match = false;
match = (invert ? !match : match);
```

Value of match being equal to true causes:

- Flexible Parser to stop and return ACCEPT
- Flexible Parser to stop and return REJECT
- Flexible Parser to set the next rule to rule specified in next_rule_name

The documentation for this struct was generated from the following file:

- fpp_ext.h

6.8 fpp_fp_table_cmd_t Struct Reference

Arguments for the FPP_CMD_FP_TABLE command.

`#include <fpp_ext.h>`

Data Fields

- uint16_t action
- uint8_t table_name [16]
- uint8_t rule_name [16]
- uint16_t position
- fpp_fp_rule_props_t r

6.8.1 Detailed Description

Arguments for the FPP_CMD_FP_TABLE command.

6.8.2 Field Documentation

6.8.2.1 action

`uint16_t action`

Action to be done

6.8.2.2 table_name

`uint8_t table_name[16]`

Name of the table to be administered by the action

6.8.2.3 rule_name

`uint8_t rule_name[16]`

Name of the rule to be added/removed to/from the table

6.8.2.4 position

`uint16_t position`

Position where to add rule (network endian)

6.8.2.5 r

`fpp_fp_rule_props_t r`

Properties of the rule - used as query result

The documentation for this struct was generated from the following file:

- fpp_ext.h

6.9 fpp_if_m_args_t Struct Reference

Match rules arguments.

`#include <fpp_ext.h>`

Data Fields

- uint16_t vlan
- uint16_t ethtype
- uint16_t sport
- uint16_t dport
- uint8_t proto
- uint8_t smac [6]
- uint8_t dmac [6]
- char fp_table0 [16]
- char fp_table1 [16]
- uint32_t hif_cookie
- struct {
  } v4
- struct {
  } v6

6.9.1 Detailed Description

Match rules arguments.

Every value corresponds to specified match rule (fpp_if_m_rules_t).

6.9.2 Field Documentation

6.9.2.1 vlan

```
uint16_t vlan
```

VLAN ID (FPP_IF_MATCH_VLAN)

Examples

fpp_cmd_log_if.c.

6.9.2.2 ethtype

```
uint16_t ethtype
```

EtherType (FPP_IF_MATCH_ETHTYPE)

6.9.2.3 sport

```
uint16_t sport
```

L4 source port number (FPP_IF_MATCH_SPORT)

6.9.2.4 dport

```
uint16_t dport
```

L4 destination port number (FPP_IF_MATCH_DPORT)

6.9.2.5 v4

```
struct { ... } v4
```

IPv4 source and destination address (FPP_IF_MATCH_SIP, FPP_IF_MATCH_DIP)

Examples

fpp_cmd_log_if.c.

6.9.2.6 v6

```
struct { ... } v6
```

IPv6 source and destination address (FPP_IF_MATCH_SIP6, FPP_IF_MATCH_DIP6)

6.9.2.7 proto

```
uint8_t proto
```

IP protocol (FPP_IF_MATCH_PROTO)

6.9.2.8 smac

`uint8_t smac[6]`

Source MAC Address (FPP_IF_MATCH_SMAC)

6.9.2.9 dmac

`uint8_t dmac[6]`

Destination MAC Address (FPP_IF_MATCH_DMAC)

6.9.2.10 fp_table0

`char fp_table0[16]`

Flexible Parser table 0 (FPP_IF_MATCH_FP0)

6.9.2.11 fp_table1

`char fp_table1[16]`

Flexible Parser table 1 (FPP_IF_MATCH_FP1)

6.9.2.12 hif_cookie

`uint32_t hif_cookie`

HIF header cookie (FPP_IF_MATCH_HIF_COOKIE)

The documentation for this struct was generated from the following file:

- fpp_ext.h

6.10 fpp_l2_bd_cmd_t Struct Reference

Data structure to be used for command buffer for L2 bridge domain control commands.

`#include <fpp_ext.h>`

Data Fields

- uint16_t action
- uint16_t vlan
- uint8_t ucast_hit
- uint8_t ucast_miss
- uint8_t mcast_hit
- uint8_t mcast_miss
- uint32_t if_list

- uint32_t untag_if_list
- fpp_l2_bd_flags_t flags

6.10.1 Detailed Description

Data structure to be used for command buffer for L2 bridge domain control commands.

It can be used:

- for command buffer in functions fci_write or fci_cmd, with commands: FPP_CMD_L2_BD.

6.10.2 Field Documentation

6.10.2.1 action

`uint16_t action`

Action to be executed (register, unregister, query, ...)

6.10.2.2 vlan

`uint16_t vlan`

VLAN ID associated with the bridge domain (network endian)

6.10.2.3 ucast_hit

`uint8_t ucast_hit`

Action to be taken when destination MAC address (uni-cast) of a packet matching the domain is found in the MAC table (network endian): 0 - Forward, 1 - Flood, 2 - Punt, 3 - Discard

6.10.2.4 ucast_miss

`uint8_t ucast_miss`

Action to be taken when destination MAC address (uni-cast) of a packet matching the domain is not found in the MAC table

6.10.2.5 mcast_hit

`uint8_t mcast_hit`

Multicast hit action

6.10.2.6 mcast_miss

```
uint8_t mcast_miss
```

Multicast miss action

6.10.2.7 if_list

```
uint32_t if_list
```

Port list (network endian). Bitmask where every set bit represents ID of physical interface being member of the domain. For instance bit (1 « 3), if set, says that interface with ID=3 is member of the domain. Only valid interface IDs are accepted by the command. If flag is set, interface is added to the domain. If flag is not set and interface has been previously added, it is removed. The IDs are given by the related FCI endpoint and related networking HW. Interface IDs can be obtained via FPP_CMD_PHY_IF.

6.10.2.8 untag_if_list

```
uint32_t untag_if_list
```

Flags marking interfaces listed in `if_list` to be 'tagged' or 'untagged' (network endian). If respective flag is set, corresponding interface within the `if_list` is treated as 'untagged' meaning that the VLAN tag will be removed. Otherwise it is configured as 'tagged'. Note that only interfaces listed within the `if_list` are taken into account.

6.10.2.9 flags

```
fpp_l2_bd_flags_t flags
```

See the fpp_l2_bd_flags_t

The documentation for this struct was generated from the following file:

- fpp_ext.h

6.11 fpp_log_if_cmd_t Struct Reference

Data structure to be used for logical interface commands.

```
#include <fpp_ext.h>
```

Data Fields

- uint16_t action
- char name [IFNAMSIZ]
- uint32_t id
- char parent_name [IFNAMSIZ]
- uint32_t parent_id

- uint32_t egress
- fpp_if_flags_t flags
- fpp_if_m_rules_t match
- fpp_if_m_args_t arguments
- fpp_algo_stats_t stats

6.11.1 Detailed Description

Data structure to be used for logical interface commands.

Usage:

- As command buffer in functions fci_write, fci_query or fci_cmd, with FPP_CMD_LOG_IF command.
- As reply buffer in functions fci_query or fci_cmd, with FPP_CMD_LOG_IF command.

Examples

fpp_cmd_log_if.c.

6.11.2 Field Documentation

6.11.2.1 action

`uint16_t action`

Action

Examples

fpp_cmd_log_if.c.

6.11.2.2 name

`char name[IFNAMSIZ]`

Interface name

Examples

fpp_cmd_log_if.c.

6.11.2.3 id

```
uint32_t id
```

Interface ID (network endian)

6.11.2.4 parent_name

```
char parent_name[IFNAMSIZ]
```

Parent physical interface name

Examples

fpp_cmd_log_if.c.

6.11.2.5 parent_id

```
uint32_t parent_id
```

Parent physical interface ID (network endian)

6.11.2.6 egress

```
uint32_t egress
```

Egress interfaces in the form of mask (to get egress id: egress & (1 < id)) must be stored in network order (network endian)

Examples

fpp_cmd_log_if.c.

6.11.2.7 flags

```
fpp_if_flags_t flags
```

Interface flags from query or flags to be set (network endian)

Examples

fpp_cmd_log_if.c.

6.11.2.8 match

`fpp_if_m_rules_t match`

Match rules from query or match rules to be set (network endian)

Examples

fpp_cmd_log_if.c.

6.11.2.9 arguments

`fpp_if_m_args_t arguments`

Arguments for match rules (network endian)

Examples

fpp_cmd_log_if.c.

6.11.2.10 stats

`fpp_algo_stats_t stats`

Logical interface statistics

The documentation for this struct was generated from the following file:

- fpp_ext.h

6.12 fpp_phy_if_cmd_t Struct Reference

Data structure to be used for physical interface commands.

```
#include <fpp_ext.h>
```

Data Fields

- uint16_t action
- char name [IFNAMSIZ]
- uint32_t id
- fpp_if_flags_t flags
- fpp_phy_if_op_mode_t mode
- fpp_phy_if_block_state_t block_state
- uint8_t mac_addr [6]
- char mirror [IFNAMSIZ]
- fpp_phy_if_stats_t stats

6.12.1 Detailed Description

Data structure to be used for physical interface commands.

Usage:

- As command buffer in functions fci_write, fci_query or fci_cmd, with FPP_CMD_PHY_IF command.
- As reply buffer in functions fci_query or fci_cmd, with FPP_CMD_PHY_IF command.

Examples

fpp_cmd_phy_if.c.

6.12.2 Field Documentation

6.12.2.1 action

```
uint16_t action
```

Action

Examples

fpp_cmd_phy_if.c.

6.12.2.2 name

```
char name[IFNAMSIZ]
```

Interface name

Examples

fpp_cmd_phy_if.c.

6.12.2.3 id

```
uint32_t id
```

Interface ID (network endian)

Examples

fpp_cmd_phy_if.c.

6.12.2.4 flags

`fpp_if_flags_t flags`

Interface flags (network endian)

Examples

fpp_cmd_phy_if.c.

6.12.2.5 mode

`fpp_phy_if_op_mode_t mode`

Phy if mode (network endian)

Examples

fpp_cmd_phy_if.c.

6.12.2.6 block_state

`fpp_phy_if_block_state_t block_state`

Phy if block state

6.12.2.7 mac_addr

`uint8_t mac_addr[6]`

Phy if MAC (network endian)

6.12.2.8 mirror

`char mirror[IFNAMSIZ]`

Name of interface to mirror the traffic to

6.12.2.9 stats

`fpp_phy_if_stats_t stats`

Physical interface statistics

The documentation for this struct was generated from the following file:

- fpp_ext.h

6.13 fpp_phy_if_stats_t Struct Reference

Physical interface statistics.

`#include <fpp_ext.h>`

Data Fields

- uint32_t ingress
- uint32_t egress
- uint32_t malformed
- uint32_t discarded

6.13.1 Detailed Description

Physical interface statistics.

Statistics used by physical interfaces (EMAC, HIF).

Note

All statistics counters are in network byte order.

6.13.2 Field Documentation

6.13.2.1 ingress

`uint32_t ingress`

Number of ingress frames for the given interface

6.13.2.2 egress

`uint32_t egress`

Number of egress frames for the given interface

6.13.2.3 malformed

`uint32_t malformed`

Number of ingress frames with detected error (i.e. checksum)

6.13.2.4 discarded

```
uint32_t discarded
```

Number of ingress frames which were discarded

The documentation for this struct was generated from the following file:

- fpp_ext.h

6.14 fpp_rt_cmd_t Struct Reference

Structure representing the command to add or remove a route.

```
#include <fpp.h>
```

Data Fields

- uint16_t action
- uint8_t dst_mac [6]
- char output_device [IFNAMSIZ]
- uint32_t id
- uint32_t flags

6.14.1 Detailed Description

Structure representing the command to add or remove a route.

Data structure to be used for command buffer for route commands. It can be used:

- as command buffer in functions fci_write, fci_query or fci_cmd, with FPP_CMD_IP_ROUTE command.
- as reply buffer in functions fci_query or fci_cmd, with FPP_CMD_IP_ROUTE command.

Examples

fpp_cmd_ip_route.c.

6.14.2 Field Documentation

6.14.2.1 action

`uint16_t action`

Action to perform

Examples

fpp_cmd_ip_route.c.

6.14.2.2 dst_mac

`uint8_t dst_mac[6]`

Destination MAC address (network endian)

Examples

fpp_cmd_ip_route.c.

6.14.2.3 output_device

`char output_device[IFNAMSIZ]`

Name of egress physical interface

Examples

fpp_cmd_ip_route.c.

6.14.2.4 id

`uint32_t id`

Unique route identifier

Examples

fpp_cmd_ip_route.c.

6.14.2.5 flags

`uint32_t flags`

Flags (network endian). 1 for IPv4 route, 2 for IPv6.

Examples

fpp_cmd_ip_route.c.

The documentation for this struct was generated from the following file:

- fpp.h

6.15 fpp_timeout_cmd_t Struct Reference

Timeout command argument.

```
#include <fpp.h>
```

6.15.1 Detailed Description

Timeout command argument.

Data structure to be used for command buffer for timeout settings. It can be used:

- for command buffer in functions fci_write, fci_query or fci_cmd, with FPP_CMD_IPV4_SET_TIMEOUT command.

The documentation for this struct was generated from the following file:

- fpp.h

Chapter 7

File Documentation

7.1 fpp.h File Reference

The legacy FCI API.

`#include <stdint.h>`

Data Structures

- struct fpp_ct_cmd_t

  Data structure used in various functions for conntrack management.
- struct fpp_ct6_cmd_t

  Data structure used in various functions for IPv6 conntrack management.
- struct fpp_rt_cmd_t

  Structure representing the command to add or remove a route.
- struct fpp_timeout_cmd_t

  Timeout command argument.

Macros

- #define FPP_ACTION_REGISTER

  Generic 'register' action for FPP_CMD_*.
- #define FPP_ACTION_DEREGISTER

  Generic 'deregister' action for FPP_CMD_*.
- #define FPP_ACTION_UPDATE

  Generic 'update' action for FPP_CMD_*.
- #define FPP_ACTION_QUERY

  Generic 'query' action for FPP_CMD_*.
- #define FPP_ACTION_QUERY_CONT

  Generic 'query continue' action for FPP_CMD_*.

- #define FPP_CMD_IPV4_CONNTRACK

  Specifies FCI command for working with IPv4 tracked connections.
- #define FPP_CMD_IPV6_CONNTRACK

  Specifies FCI command for working with IPv6 tracked connections.
- #define FPP_CMD_IP_ROUTE

  Specifies FCI command for working with routes.
- #define FPP_CMD_IPV4_RESET

  Specifies FCI command that clears all IPv4 routes (see FPP_CMD_IP_ROUTE) and conntracks (see FPP_CMD_IPV4_CONNTRACK)
- #define FPP_CMD_IPV6_RESET

  Specifies FCI command that clears all IPv6 routes (see FPP_CMD_IP_ROUTE) and conntracks (see FPP_CMD_IPV6_CONNTRACK)
- #define FPP_CMD_IPV4_SET_TIMEOUT

  Specifies FCI command for setting timeouts of conntracks.

7.1.1 Detailed Description

The legacy FCI API.

This file origin is the fpp.h file from CMM sources.

7.1.2 Macro Definition Documentation

7.1.2.1 FPP_CMD_IPV4_CONNTRACK

```
#define FPP_CMD_IPV4_CONNTRACK
```

Specifies FCI command for working with IPv4 tracked connections.

This command can be used with various values of `.action`:

- `FPP_ACTION_REGISTER`: Defines a connection and binds it to previously created route(s).
- `FPP_ACTION_DEREGISTER`: Deletes previously defined connection.
- `FPP_ACTION_QUERY`: Gets parameters of existing connection. It creates a snapshot of all active conntrack entries and replies with first of them.
- `FPP_ACTION_QUERY_CONT`: Shall be called periodically after `FPP_ACTION_QUERY` was called. On each call it replies with parameters of next connection. It returns `FPP_ERR_CT_ENTRY_NOT_FOUND` when no more entries exist.

Command Argument Type: fpp_ct_cmd_t

Action FPP_ACTION_REGISTER

Items to be set in command argument structure:

```
fpp_ct_cmd_t cmd_data =
{
  // Register new conntrack
  .action = FPP_ACTION_REGISTER,
  // Source IPv4 address (network endian)
  .saddr = ...,
  // Destination IPv4 address (network endian)
  .daddr = ...,
  // Source port (network endian)
  .sport = ...,
  // Destination port (network endian)
  .dport = ...,
  // Reply source IPv4 address (network endian). Used for NAT, otherwise equals .daddr
  .saddr_reply = ...,
  // Reply destination IPv4 address (network endian). Used for NAT, otherwise equals .saddr
  .daddr_reply = ...,
  // Reply source port (network endian). Used for NAT, otherwise equals .dport
  .sport_reply = ...,
  // Reply destination port (network endian). Used for NAT, otherwise equals .sport
  .dport_reply = ...,
  // IP protocol ID (17=UDP, 6=TCP, ...)
  .protocol = ...,
  // Bidirectional/Single direction (network endian)
  .flags = ...,
  // ID of route previously created with .FPP_CMD_IP_ROUTE command (network endian)
  .route_id = ...,
  // ID of reply route previously created with .FPP_CMD_IP_ROUTE command (network endian)
  .route_id_reply = ...
};
```

By default the connection is created as bi-directional. It means that two routing table entries are created at once: one for standard flow given by .saddr, .daddr, .sport, .dport, and .protocol and one for reverse flow defined by .saddr_reply, .daddr_reply, .sport_reply and .dport_reply. To create single-directional connection, either:

- set `.flags |= CTCMD_FLAGS_REP_DISABLED` and don't set `route_id_reply`, or
- set `.flags |= CTCMD_FLAGS_ORIG_DISABLED` and don't set `route_id`.

To configure NAT-ed connection, set reply addresses and/or ports different than original addresses and ports. To achieve NAPT (also called PAT), use `daddr_reply`, `dport_reply`, `saddr_reply`, and `sport_reply`:

1. `daddr_reply != saddr`: Source address of packets in original direction will be changed from `saddr` to `daddr_reply`. In case of bi-directional connection, destination address of packets in reply direction will be changed from `daddr_reply` to `saddr`.
2. `dport_reply != sport`: Source port of packets in original direction will be changed from `sport` to `dport_reply`. In case of bi-directional connection, destination port of packets in reply direction will be changed from `dport_reply` to `sport`.
3. `saddr_reply != daddr`: Destination address of packets in original direction will be changed from `daddr` to `saddr_reply`. In case of bi-directional connection, source address of packets in reply direction will be changed from `saddr_reply` to `daddr`.

4. `sport_reply != dport`: Destination port of packets in original direction will be changed from `dport` to `sport_reply`. In case of bi-directional connection, source port of packets in reply direction will be changed from `sport_reply` to `dport`.

Action FPP_ACTION_DEREGISTER

Items to be set in command argument structure:

```
fpp_ct_cmd_t cmd_data =
{
  .action = FPP_ACTION_DEREGISTER, // Deregister previously created conntrack
  .saddr = ...,                    // Source IPv4 address (network endian)
  .daddr = ...,                    // Destination IPv4 address (network endian)
  .sport = ...,                    // Source port (network endian)
  .dport = ...,                    // Destination port (network endian)
  .saddr_reply = ...,              // Reply source IPv4 address (network endian)
  .daddr_reply = ...,              // Reply destination IPv4 address (network endian)
  .sport_reply = ...,              // Reply source port (network endian)
  .dport_reply = ...,              // Reply destination port (network endian)
  .protocol = ...,                 // IP protocol ID
};
```

Action FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Items to be set in command argument structure:

```
fpp_ct_cmd_t cmd_data =
{
  .action = ...     // Either FPP_ACTION_QUERY or FPP_ACTION_QUERY_CONT
};
```

Response data type for queries: fpp_ct_cmd_t

Response data provided:

```
rsp_data.saddr;         // Source IPv4 address (network endian)
rsp_data.daddr;         // Destination IPv4 address (network endian)
rsp_data.sport;         // Source port (network endian)
rsp_data.dport;         // Destination port (network endian)
rsp_data.saddr_reply;   // Reply source IPv4 address (network endian)
rsp_data.daddr_reply;   // Reply destination IPv4 address (network endian)
rsp_data.sport_reply;   // Reply source port (network endian)
rsp_data.dport_reply;   // Reply destination port (network endian)
rsp_data.protocol;      // IP protocol ID (17=UDP, 6=TCP, ...)
```

Examples

fpp_cmd_ipv4_conntrack.c.

7.1.2.2 FPP_CMD_IPV6_CONNTRACK

`#define FPP_CMD_IPV6_CONNTRACK`

Specifies FCI command for working with IPv6 tracked connections.

This command can be used with various values of `.action`:

- `FPP_ACTION_REGISTER`: Defines a connection and binds it to previously created route(s).
- `FPP_ACTION_DEREGISTER`: Deletes previously defined connection.

- FPP_ACTION_QUERY: Gets parameters of existing connection. It creates a snapshot of all active conntrack entries and replies with first of them.
- FPP_ACTION_QUERY_CONT: Shall be called periodically after FPP_ACTION_QUERY was called. On each call it replies with parameters of next connection. It returns FPP_ERR_CT_ENTRY_NOT_FOUND when no more entries exist.

Command Argument Type: fpp_ct6_cmd_t

Action FPP_ACTION_REGISTER

Items to be set in command argument structure:

```
fpp_ct6_cmd_t cmd_data =
{
  // Register new conntrack
  .action = FPP_ACTION_REGISTER,
  // Source IPv6 address, (network endian)
  .saddr[0..3] = ...,
  // Destination IPv6 address, (network endian)
  .daddr[0..3] = ...,
  // Source port (network endian)
  .sport = ...,
  // Destination port (network endian)
  .dport = ...,
  // Reply source IPv6 address (network endian). Used for NAT, otherwise equals .daddr
  .saddr_reply[0..3] = ...,
  // Reply destination IPv6 address (network endian). Used for NAT, otherwise equals .saddr
  .daddr_reply[0..3] = ...,
  // Reply source port (network endian). Used for NAT, otherwise equals .dport
  .sport_reply = ...,
  // Reply destination port (network endian). Used for NAT, otherwise equals .sport
  .dport_reply = ...,
  // IP protocol ID (17=UDP, 6=TCP, ...)
  .protocol = ...,
  // Bidirectional/Single direction (network endian)
  .flags = ...,
  // ID of route previously created with .FPP_CMD_IP_ROUTE command (network endian)
  .route_id = ...,
  // ID of reply route previously created with .FPP_CMD_IP_ROUTE command (network endian)
  .route_id_reply = ...
};
```

By default the connection is created as bi-directional. It means that two routing table entries are created at once: one for standard flow given by .saddr, .daddr, .sport, .dport, and .protocol and one for reverse flow defined by .saddr_reply, .daddr_reply, .sport_reply and .dport_reply. To create single-directional connection, either:

- set .flags |= CTCMD_FLAGS_REP_DISABLED and don't set route_id_reply, or
- set .flags |= CTCMD_FLAGS_ORIG_DISABLED and don't set route_id.

To configure NAT-ed connection, set reply addresses and/or ports different than original addresses and ports. To achieve NAPT (also called PAT), use daddr_reply, dport_reply, saddr_reply, and sport_reply:

1. daddr_reply != saddr: Source address of packets in original direction will be changed from saddr to daddr_reply. In case of bi-directional connection,

destination address of packets in reply direction will be changed from `daddr_reply` to `saddr`.

2. `dport_reply != sport`: Source port of packets in original direction will be changed from `sport` to `dport_reply`. In case of bi-directional connection, destination port of packets in reply direction will be changed from `dport_reply` to `sport`.
3. `saddr_reply != daddr`: Destination address of packets in original direction will be changed from `daddr` to `saddr_reply`. In case of bi-directional connection, source address of packets in reply direction will be changed from `saddr_reply` to `daddr`.
4. `sport_reply != dport`: Destination port of packets in original direction will be changed from `dport` to `sport_reply`. In case of bi-directional connection, source port of packets in reply direction will be changed from `sport_reply` to `dport`.

Action FPP_ACTION_DEREGISTER

Items to be set in command argument structure:

```
fpp_ct6_cmd_t cmd_data =
{
  .action = FPP_ACTION_DEREGISTER, // Deregister previously created conntrack
  .saddr[0..3] = ...,              // Source IPv6 address, (network endian)
  .daddr[0..3] = ...,              // Destination IPv6 address, (network endian)
  .sport = ...,                    // Source port (network endian)
  .dport = ...,                    // Destination port (network endian)
  .saddr_reply[0..3] = ...,        // Reply source IPv6 address (network endian)
  .daddr_reply[0..3] = ...,        // Reply destination IPv6 address (network endian)
  .sport_reply = ...,              // Reply source port (network endian)
  .dport_reply = ...,              // Reply destination port (network endian)
  .protocol = ...,                 // IP protocol ID
};
```

Action FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Items to be set in command argument structure:

```
fpp_ct6_cmd_t cmd_data =
{
  .action = ...      // Either FPP_ACTION_QUERY or FPP_ACTION_QUERY_CONT
};
```

Response data type for queries: fpp_ct6_cmd_t

Response data provided (all values in network byte order):

```
rsp_data.saddr;          // Source IPv6 address (network endian)
rsp_data.daddr;          // Destination IPv6 address (network endian)
rsp_data.sport;          // Source port (network endian)
rsp_data.dport;          // Destination port (network endian)
rsp_data.saddr_reply;    // Reply source IPv6 address (network endian)
rsp_data.daddr_reply;    // Reply destination IPv6 address (network endian)
rsp_data.sport_reply;    // Reply source port (network endian)
rsp_data.dport_reply;    // Reply destination port (network endian)
rsp_data.protocol;       // IP protocol ID (17=UDP, 6=TCP, ...)
```

Examples

fpp_cmd_ipv6_conntrack.c.

7.1.2.3 FPP_CMD_IP_ROUTE

```
#define FPP_CMD_IP_ROUTE
```

Specifies FCI command for working with routes.

Routes are representing direction where matching traffic shall be forwarded to. Every route specifies egress physical interface and MAC address of next network node. This command can be used with various values of `.action`:

- `FPP_ACTION_REGISTER`: Defines a new route.
- `FPP_ACTION_DEREGISTER`: Deletes previously defined route.
- `FPP_ACTION_QUERY`: Gets parameters of existing routes. It creates a snapshot of all active route entries and replies with first of them.
- `FPP_ACTION_QUERY_CONT`: Shall be called periodically after `FPP_ACTION_QUERY` was called. On each call it replies with parameters of next route. It returns `FPP_ERR_RT_ENTRY_NOT_FOUND` when no more entries exist.

Command Argument Type: fpp_rt_cmd_t

Action FPP_ACTION_REGISTER

Items to be set in command argument structure:

```
fpp_rt_cmd_t cmd_data =
{
  .action = FPP_ACTION_REGISTER, // Register new route
  .dst_mac = ...,                // Destination MAC address (network endian)
  .output_device = ...,          // Name of egress interface (name of physical interface)
  .id = ...                      // Chosen number will be used as unique route identifier
        (network endian)
  .flags = ...,                  // 1 for IPv4 addressing, 2 for IPv6 (network endian)
};
```

Action FPP_ACTION_DEREGISTER

Items to be set in command argument structure:

```
fpp_rt_cmd_t cmd_data =
{
  .action = FPP_ACTION_DEREGISTER, // Deregister a route
  .id = ...                        // Unique route identifier (network endian)
};
```

Action FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Items to be set in command argument structure:

```
fpp_rt_cmd_t cmd_data =
{
  .action = ...      // Either FPP_ACTION_QUERY or FPP_ACTION_QUERY_CONT
};
```

Response data provided (fpp_rt_cmd_t):

```
rsp_data.dst_mac;         // Destination MAC address
rsp_data.output_device;   // Output device name
rsp_data.id;              // Route ID (network endian)
srp_data.flags;           // Flags (network endian)
```

Examples

fpp_cmd_ip_route.c.

7.1.2.4 FPP_CMD_IPV4_RESET

```
#define FPP_CMD_IPV4_RESET
```

Specifies FCI command that clears all IPv4 routes (see FPP_CMD_IP_ROUTE) and conntracks (see FPP_CMD_IPV4_CONNTRACK)

This command uses no arguments.

Command Argument Type: none (cmd_buf = NULL; cmd_len = 0;)

Examples

fpp_cmd_ip_route.c.

7.1.2.5 FPP_CMD_IPV6_RESET

```
#define FPP_CMD_IPV6_RESET
```

Specifies FCI command that clears all IPv6 routes (see FPP_CMD_IP_ROUTE) and conntracks (see FPP_CMD_IPV6_CONNTRACK)

This command uses no arguments.

Command Argument Type: none (cmd_buf = NULL; cmd_len = 0;)

Examples

fpp_cmd_ip_route.c.

7.1.2.6 FPP_CMD_IPV4_SET_TIMEOUT

```
#define FPP_CMD_IPV4_SET_TIMEOUT
```

Specifies FCI command for setting timeouts of conntracks.

This command sets timeout for conntracks based on protocol. Three kinds of protocols are distinguished: TCP, UDP and others. For each of them timeout can be set independently. For UDP it is possible to set different value for bidirectional and single-directional connection. Default timeout value is 5 days for TCP, 300s for UDP and 240s for others.

Newly created connections are being created with new timeout values already set. Previously created connections have their timeout updated with first received packet.

Command Argument Type: fpp_timeout_cmd_t

Items to be set in command argument structure:
```
fpp_timeout_cmd_t cmd_data =
{
  // IP protocol to be affected. Either 17 for UDP, 6 for TCP or 0 for others.
  .protocol;
  // Use 0 for normal connections, 1 for 4over6 IP tunnel connections.
  .sam_4o6_timeout;
  // Timeout value in seconds.
  .timeout_value1;
  // Optional timeout value which is valid only for UDP connections. If the value is set
  // (non zero), then it affects unidirectional UDP connections only.
  .timeout_value2;
};
```

7.2 fpp_ext.h File Reference

Extension of the legacy fpp.h.

Data Structures

- struct fpp_if_m_args_t

  Match rules arguments.
- struct fpp_phy_if_stats_t

  Physical interface statistics.
- struct fpp_algo_stats_t

  Algorithm statistics.
- struct fpp_phy_if_cmd_t

  Data structure to be used for physical interface commands.
- struct fpp_log_if_cmd_t

  Data structure to be used for logical interface commands.
- struct fpp_l2_bd_cmd_t

  Data structure to be used for command buffer for L2 bridge domain control commands.
- struct fpp_fp_rule_props_t

  Properties of the Flexible parser rule.
- struct fpp_fp_rule_cmd_t

  Arguments for the FPP_CMD_FP_RULE command.
- struct fpp_fp_table_cmd_t

  Arguments for the FPP_CMD_FP_TABLE command.
- struct fpp_buf_cmd_t

  Argument structure for the FPP_CMD_DATA_BUF_PUT command.

Macros

- #define FPP_CMD_PHY_IF

  FCI command for working with physical interfaces.

- #define FPP_CMD_LOG_IF

    FCI command for working with logical interfaces.
- #define FPP_CMD_IF_LOCK_SESSION

    FCI command to perform lock on interface database.
- #define FPP_CMD_IF_UNLOCK_SESSION

    FCI command to perform unlock on interface database.
- #define FPP_CMD_L2_BD

    VLAN-based L2 bridge domain management.
- #define FPP_CMD_FP_TABLE

    Administers the Flexible Parser tables.
- #define FPP_CMD_FP_RULE

    Administers the Flexible Parser rules.
- #define FPP_ACTION_USE_RULE

    Flexible Parser specific 'use' action for FPP_CMD_FP_TABLE.
- #define FPP_ACTION_UNUSE_RULE

    Flexible Parser specific 'unuse' action for FPP_CMD_FP_TABLE.
- #define FPP_CMD_FP_FLEXIBLE_FILTER

    Uses flexible parser to filter out frames from further processing.
- #define FPP_CMD_DATA_BUF_PUT

    FCI command to send an arbitrary data to the accelerator.
- #define FPP_CMD_DATA_BUF_AVAIL

    Event reported when accelerator wants to send a data buffer to host.

Enumerations

- enum fpp_if_flags_t {
  FPP_IF_ENABLED, FPP_IF_PROMISC,
  FPP_IF_MATCH_OR, FPP_IF_DISCARD,
  FPP_IF_MIRROR }

    Interface flags.
- enum fpp_phy_if_op_mode_t {
  FPP_IF_OP_DISABLED, FPP_IF_OP_DEFAULT,
  FPP_IF_OP_BRIDGE, FPP_IF_OP_ROUTER,
  FPP_IF_OP_VLAN_BRIDGE, FPP_IF_OP_FLEXIBLE_ROUTER }

    Physical if modes.
- enum fpp_if_m_rules_t {
  FPP_IF_MATCH_TYPE_ETH, FPP_IF_MATCH_TYPE_VLAN,
  FPP_IF_MATCH_TYPE_PPPOE, FPP_IF_MATCH_TYPE_ARP,
  FPP_IF_MATCH_TYPE_MCAST, FPP_IF_MATCH_TYPE_IPV4,
  FPP_IF_MATCH_TYPE_IPV6, FPP_IF_MATCH_RESERVED7,
  FPP_IF_MATCH_RESERVED8, FPP_IF_MATCH_TYPE_IPX,
  FPP_IF_MATCH_TYPE_BCAST, FPP_IF_MATCH_TYPE_UDP,

FPP_IF_MATCH_TYPE_TCP, FPP_IF_MATCH_TYPE_ICMP, FPP_IF_MATCH_TYPE_IGMP, FPP_IF_MATCH_VLAN, FPP_IF_MATCH_PROTO, FPP_IF_MATCH_SPORT, FPP_IF_MATCH_DPORT, FPP_IF_MATCH_SIP6, FPP_IF_MATCH_DIP6, FPP_IF_MATCH_SIP, FPP_IF_MATCH_DIP, FPP_IF_MATCH_ETHTYPE, FPP_IF_MATCH_FP0, FPP_IF_MATCH_FP1, FPP_IF_MATCH_SMAC, FPP_IF_MATCH_DMAC, FPP_IF_MATCH_HIF_COOKIE }

  Match rules. Can be combined using bitwise OR.

- enum fpp_phy_if_block_state_t { BS_NORMAL, BS_BLOCKED, BS_LEARN_ONLY, BS_FORWARD_ONLY }

  Interface blocking state.

- enum fpp_l2_bd_flags_t { FPP_L2BR_DOMAIN_DEFAULT, FPP_L2BR_DOMAIN_FALLBACK }

  L2 bridge domain flags.

- enum fpp_fp_rule_match_action_t { FP_ACCEPT, FP_REJECT, FP_NEXT_RULE }

  Specifies the Flexible Parser result on the rule match.

- enum fpp_fp_offset_from_t { FP_OFFSET_FROM_L2_HEADER, FP_OFFSET_FROM_L3_HEADER, FP_OFFSET_FROM_L4_HEADER }

  Specifies how to calculate the frame data offset.

7.2.1 Detailed Description

Extension of the legacy fpp.h.

All FCI commands and related elements not present within the legacy fpp.h shall be put into this file. All macro values (uint16_t) shall have the upper nibble set to b1111 to ensure no conflicts with the legacy macro values.

Note

  Documentation is part of libfci.h.

7.3 libfci.h File Reference

Generic LibFCI header file.

Macros

- #define FCI_CFG_FORCE_LEGACY_API

Changes the LibFCI API so it is more compatible with legacy implementation.

- #define FPP_CMD_IPV4_CONNTRACK_CHANGE
- #define FPP_CMD_IPV6_CONNTRACK_CHANGE
- #define CTCMD_FLAGS_ORIG_DISABLED

  Disable connection originator.
- #define CTCMD_FLAGS_REP_DISABLED

  Disable connection replier.

Enumerations

- enum fci_mcast_groups_t { FCI_GROUP_NONE, FCI_GROUP_CATCH }

  List of supported multicast groups.
- enum fci_client_type_t { FCI_CLIENT_DEFAULT }

  List of supported FCI client types.
- enum fci_cb_retval_t { FCI_CB_STOP, FCI_CB_CONTINUE }

  The FCI callback return values.

Functions

- FCI_CLIENT * fci_open (fci_client_type_t type, fci_mcast_groups_t group)

  Creates new FCI client and opens a connection to FCI endpoint.
- int fci_close (FCI_CLIENT *client)

  Disconnects from FCI endpoint and destroys FCI client instance.
- int fci_catch (FCI_CLIENT *client)

  Catch and process all FCI messages delivered to the FCI client.
- int fci_cmd (FCI_CLIENT *client, unsigned short fcode, unsigned short *cmd_buf, unsigned short cmd_len, unsigned short *rep_buf, unsigned short *rep_len)

  Run an FCI command with optional data response.
- int fci_query (FCI_CLIENT *this_client, unsigned short fcode, unsigned short cmd_len, unsigned short *pcmd, unsigned short *rsplen, unsigned short *rsp_data)

  Run an FCI command with data response.
- int fci_write (FCI_CLIENT *client, unsigned short fcode, unsigned short cmd_len, unsigned short *cmd_buf)

  Run an FCI command.
- int fci_register_cb (FCI_CLIENT *client, fci_cb_retval_t(*event_cb)(unsigned short fcode, unsigned short len, unsigned short *payload))

  Register event callback function.
- int fci_fd (FCI_CLIENT *this_client)

  Obsolete function, shall not be used.

7.3.1 Detailed Description

Generic LibFCI header file.

This file contains generic API and API description

Chapter 8

Example Documentation

8.1 fpp_cmd_ip_route.c

```
/* =========================================================================
 *  Copyright 2020 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <errno.h>
#include "libfci.h"
#include "fpp.h"
#include "fpp_ext.h"
#include "fci_examples.h"
/*
 * @brief       Reset IPv4 and IPv6 router
 * @param[in]   cl The FCI client instance
 */
void fci_router_reset(FCI_CLIENT *cl)
{
    int ret;
    ret = fci_write(cl, FPP_CMD_IPV4_RESET, 0, NULL);
    if (FPP_ERR_OK != ret)
    {
        printf("FPP_CMD_IPV4_RESET failed: %d\n", ret);
    }
```

```
    ret = fci_write(cl, FPP_CMD_IPV6_RESET, 0, NULL);
    if (FPP_ERR_OK != ret)
    {
        printf("FPP_CMD_IPV6_RESET failed: %d\n", ret);
    }
}
/*
 * @brief       Register two IPv4 routes
 * @details     Function registers two IPv4 routes: one targeting emac0 and the second
 *              one emac1. Traffic matching respective route will be forwarded via
 *              physical interface given by 'fpp_rt_cmd_t.output_device' while its source
 *              MAC address will be replaced by MAC address of the output interface and
 *              destination MAC address will be replaced by 'fpp_rt_cmd_t.dst_mac'.
 * @param[in]   cl The FCI client instance
 */
void fci_router_register_ipv4_routes(FCI_CLIENT *cl)
{
    int ret;
    fpp_rt_cmd_t r1 =
    {
        /*  Register new route */
        .action = FPP_ACTION_REGISTER,
        /*  Destination MAC address to be used for packets matching the route */
        .dst_mac = {0x00, 0xaa, 0xbb, 0xcc, 0xdd, 0xee},
        /*  Egress physical interface */
        .output_device = "emac0",
        /*  Unique route identifier */
        .id = htonl(123),
        /*  Use IPv4 addressing */
        .flags = htonl(1)
    };
    fpp_rt_cmd_t r2 =
    {
        .action = FPP_ACTION_REGISTER,
        .dst_mac = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55},
        .output_device = "emac1",
        .id = htonl(456),
        .flags = htonl(1),
    };
    /*  Register route "r1" */
    ret = fci_write(cl, FPP_CMD_IP_ROUTE, sizeof(r1), (void *)&r1);
    if (FPP_ERR_OK != ret)
    {
        printf("FPP_CMD_IP_ROUTE[FPP_ACTION_REGISTER] failed: %d\n", ret);
        return;
    }
    /*  Register route "r2" */
    ret = fci_write(cl, FPP_CMD_IP_ROUTE, sizeof(r2), (void *)&r2);
    if (FPP_ERR_OK != ret)
    {
        printf("FPP_CMD_IP_ROUTE[FPP_ACTION_REGISTER] failed: %d\n", ret);
        return;
    }
}
/*
 * @brief       Register two IPv6 routes
 * @details     Function registers two IPv6 routes: one targeting emac0 and the second
 *              one emac1. Traffic matching respective route will be forwarded via
 *              physical interface given by 'fpp_rt_cmd_t.output_device' while its source
 *              MAC address will be replaced by MAC address of the output interface and
 *              destination MAC address will be replaced by 'fpp_rt_cmd_t.dst_mac'.
 * @param[in]   cl The FCI client instance
 */
void fci_router_register_ipv6_routes(FCI_CLIENT *cl)
{
    int ret;
    fpp_rt_cmd_t r1 =
    {
        /*  Register new route */
        .action = FPP_ACTION_REGISTER,
        /*  Destination MAC address to be used for packets matching the route */
        .dst_mac = {0x00, 0xaa, 0xbb, 0xcc, 0xdd, 0xee},
        /*  Egress physical interface */
```

```
        .output_device = "emac0",
        /*  Unique route identifier */
        .id = htonl(111),
        /*  Use IPv6 addressing */
        .flags = htonl(2)
    };
    fpp_rt_cmd_t r2 =
    {
        .action = FPP_ACTION_REGISTER,
        .dst_mac = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55},
        .output_device = "emac1",
        .id = htonl(222),
        .flags = htonl(2),
    };
    /*  Register route "r1" */
    ret = fci_write(cl, FPP_CMD_IP_ROUTE, sizeof(r1), (void *)&r1);
    if (FPP_ERR_OK != ret)
    {
        printf("FPP_CMD_IP_ROUTE[FPP_ACTION_REGISTER] failed: %d\n", ret);
        return;
    }
    /*  Register route "r2" */
    ret = fci_write(cl, FPP_CMD_IP_ROUTE, sizeof(r2), (void *)&r2);
    if (FPP_ERR_OK != ret)
    {
        printf("FPP_CMD_IP_ROUTE[FPP_ACTION_REGISTER] failed: %d\n", ret);
        return;
    }
}
/*
 * @brief       Print all IPv4 routes
 * @param[in]   cl The FCI client instance
 */
void fci_router_print_ipv4_routes(FCI_CLIENT *cl)
{
    int ret;
    fpp_rt_cmd_t cmd;
    fpp_rt_cmd_t rep;
    unsigned short rep_len;
    cmd.action = FPP_ACTION_QUERY;
    ret = fci_query(cl, FPP_CMD_IP_ROUTE, sizeof(cmd), (void *)&cmd, &rep_len, (void *)&rep);
    while (FPP_ERR_OK == ret)
    {
        if (1 == ntohl(rep.flags))
        {
            printf("%03d: %s (%02x:%02x:%02x:%02x:%02x:%02x), flags: 0x%x\n",
                ntohl(rep.id), rep.output_device, rep.dst_mac[0], rep.dst_mac[1],
                    rep.dst_mac[2], rep.dst_mac[3], rep.dst_mac[4], rep.dst_mac[5],
                        ntohl(rep.flags));
        }
        cmd.action = FPP_ACTION_QUERY_CONT;
        ret = fci_query(cl, FPP_CMD_IP_ROUTE, sizeof(cmd), (void *)&cmd, &rep_len, (void
       *)&rep);
    }
}
/*
 * @brief       Print all IPv6 routes
 * @param[in]   cl The FCI client instance
 */
void fci_router_print_ipv6_routes(FCI_CLIENT *cl)
{
    int ret;
    fpp_rt_cmd_t cmd;
    fpp_rt_cmd_t rep;
    unsigned short rep_len;
    cmd.action = FPP_ACTION_QUERY;
    ret = fci_query(cl, FPP_CMD_IP_ROUTE, sizeof(cmd), (void *)&cmd, &rep_len, (void *)&rep);
    while (FPP_ERR_OK == ret)
    {
        if (2 == ntohl(rep.flags))
        {
            printf("%03d: %s (%02x:%02x:%02x:%02x:%02x:%02x), flags: 0x%x\n",
                ntohl(rep.id), rep.output_device, rep.dst_mac[0], rep.dst_mac[1],
```

```
                    rep.dst_mac[2], rep.dst_mac[3], rep.dst_mac[4], rep.dst_mac[5],
                        ntohl(rep.flags));
            }
            cmd.action = FPP_ACTION_QUERY_CONT;
            ret = fci_query(cl, FPP_CMD_IP_ROUTE, sizeof(cmd), (void *)&cmd, &rep_len, (void
           *)&rep);
        }
}
/*
 * @brief       Remove routes created by fci_router_register_ipv4_routes()
 * @param[in]   cl The FCI client instance
 */
void fci_router_remove_ipv4_routes(FCI_CLIENT *cl)
{
    int ret;
    fpp_rt_cmd_t cmd;
    cmd.action = FPP_ACTION_DEREGISTER;
    cmd.id = htonl(123);
    ret = fci_write(cl, FPP_CMD_IP_ROUTE, sizeof(cmd), (void *)&cmd);
    if (FPP_ERR_OK != ret)
    {
        printf("FPP_CMD_IP_ROUTE[FPP_ACTION_DEREGISTER] failed: %d\n", ret);
    }
    cmd.id = htonl(456);
    ret = fci_write(cl, FPP_CMD_IP_ROUTE, sizeof(cmd), (void *)&cmd);
    if (FPP_ERR_OK != ret)
    {
        printf("FPP_CMD_IP_ROUTE[FPP_ACTION_DEREGISTER] failed: %d\n", ret);
    }
}
/*
 * @brief       Remove routes created by fci_router_register_ipv6_routes()
 * @param[in]   cl The FCI client instance
 */
void fci_router_remove_ipv6_routes(FCI_CLIENT *cl)
{
    int ret;
    fpp_rt_cmd_t cmd;
    cmd.action = FPP_ACTION_DEREGISTER;
    cmd.id = htonl(111);
    ret = fci_write(cl, FPP_CMD_IP_ROUTE, sizeof(cmd), (void *)&cmd);
    if (FPP_ERR_OK != ret)
    {
        printf("FPP_CMD_IP_ROUTE[FPP_ACTION_DEREGISTER] failed: %d\n", ret);
    }
    cmd.id = htonl(222);
    ret = fci_write(cl, FPP_CMD_IP_ROUTE, sizeof(cmd), (void *)&cmd);
    if (FPP_ERR_OK != ret)
    {
        printf("FPP_CMD_IP_ROUTE[FPP_ACTION_DEREGISTER] failed: %d\n", ret);
    }
}
```

8.2 fpp_cmd_ipv4_conntrack.c

```
/* =========================================================================
 *  Copyright 2020 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
```

```
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <errno.h>
#include "libfci.h"
#include "fpp.h"
#include "fpp_ext.h"
#include "fci_examples.h"
/*
 * @brief       Register two UPD connections to be fast-forwarded
 * @details     Add 2 routing table entries (conntracks). Traffic matching
 *              respective conntrack will be forwarded via physical interface
 *              given by matching route (fpp_rt_cmd_t.output_device) while its
 *              source MAC address will be replaced by MAC address of the output
 *              interface and destination MAC address will be replaced by the
 *              one defined by route (fpp_rt_cmd_t.dst_mac).
 *
 *              In case of no hit, packet will be sent to default logical
 *              interface (to host). Host can configure slow-path routing using
 *              standard OS-provided mechanisms to route rest of traffic (e.g.
 *              ICMP).
 * @param[in]   cl The FCI client instance
 */
void fci_router_register_ipv4_conntracks(FCI_CLIENT *cl)
{
    int ret;
    fpp_ct_cmd_t ct1 =
    {
        /*  New connection */
        .action = FPP_ACTION_REGISTER,
        /*  Source IP address: 11.41.48.100 */
        .saddr = htonl(0x0b293064),
        /*  Destination IP address: 12.41.48.100 */
        .daddr = htonl(0x0c293064),
        /*  Source L4 port */
        .sport = htons(11),
        /*  Destination L4 port */
        .dport = htons(12),
        /*  Source IP address in reply direction. Equal to 'daddr' to disable
            replacement. */
        .saddr_reply = htonl(0x0c293064),
        /*  Destination IP address in reply direction. Same as 'saddr' to
            disable replacement. */
        .daddr_reply = htonl(0x0b293064),
        /*  Source L4 port in reply direction. Equal to 'dport' to disable
            replacement. */
        .sport_reply = htons(12),
        /*  Destination L4 port in reply direction. Equal to 'sport' to disable
            replacement.*/
        .dport_reply = htons(11),
        /*  Protocol ID: UDP */
        .protocol = 17,
        /*  Flags: Do not open reply connection */
        .flags = htons(CTCMD_FLAGS_REP_DISABLED),
        /*  Associated route (456=emac1). This route will be used to forward
            packets matching this tracked connection (SIP+DIP+SPORT+DPORT+PROTO).
            Route must exist. To create a route see the FPP_CMD_IP_ROUTE. */
        .route_id = htonl(456),
    };
```

```
    fpp_ct_cmd_t ct2 =
    {
        .action = FPP_ACTION_REGISTER,
        /* Source IP address: 12.41.48.100 */
        .saddr = htonl(0x0c293064),
        /* Destination IP address: 11.41.48.100 */
        .daddr = htonl(0x0b293064),
        .sport = htons(12),
        .dport = htons(11),
        .saddr_reply = htonl(0x0b293064),
        .daddr_reply = htonl(0x0c293064),
        .sport_reply = htons(11),
        .dport_reply = htons(12),
        .protocol = 17,
        .flags = htons(CTCMD_FLAGS_REP_DISABLED),
        .route_id = htonl(123),
    };
    /* Register connection "ct1" */
    ret = fci_write(cl, FPP_CMD_IPV4_CONNTRACK, sizeof(ct1), (void *)&ct1);
    if (0 != ret)
    {
        printf("FPP_CMD_IPV4_CONNTRACK[FPP_ACTION_REGISTER] failed: %d\n", ret);
        return;
    }
    /* Register connection "ct2" */
    ret = fci_write(cl, FPP_CMD_IPV4_CONNTRACK, sizeof(ct2), (void *)&ct2);
    if (0 != ret)
    {
        printf("FPP_CMD_IPV4_CONNTRACK[FPP_ACTION_REGISTER] failed: %d\n", ret);
        return;
    }
}
/*
 * @brief       Register two UPD connections to be fast-forwarded with NAT
 * @details     Add 2 routing table entries (conntracks). Traffic matching
 *              respective conntrack will be forwarded via physical interface
 *              given by matching route (fpp_rt_cmd_t.output_device) while its
 *              source MAC address will be replaced by MAC address of the output
 *              interface and destination MAC address will be replaced by the
 *              one defined by route (fpp_rt_cmd_t.dst_mac). Additionally, source
 *              and destination IP address will be replaced using '.daddr_reply'
 *              and '.saddr_reply' as well as source and destination port number
 *              will be replaced by '.dport_reply' and '.sport_reply' values.
 *
 *              In case of no hit, packet will be sent to default logical
 *              interface (to host). Host can configure slow-path routing using
 *              standard OS-provided mechanisms to route rest of traffic (e.g.
 *              ICMP).
 * @param[in]   cl The FCI client instance
 */
void fci_router_register_ipv4_conntracks_nat(FCI_CLIENT *cl)
{
    int ret;
    fpp_ct_cmd_t ct1 =
    {
        /* New connection */
        .action = FPP_ACTION_REGISTER,
        /* Source IP address: 11.41.48.100 */
        .saddr = htonl(0x0b293064),
        /* Destination IP address: 12.41.48.100 */
        .daddr = htonl(0x0c293064),
        /* Source L4 port */
        .sport = htons(11),
        /* Destination L4 port */
        .dport = htons(12),
        /* Source IP address in reply direction. Destination IP address of
           routed packet will be replaced by this value (120.41.48.100). */
        .saddr_reply = htonl(0x78293064),
        /* Destination IP address in reply direction. Source IP address of
           routed packet will be replaced by this value (110.41.48.100). */
        .daddr_reply = htonl(0x6e293064),
        /* Source L4 port in reply direction. Destination port of routed packet
           will be replaced by this value. */
```

```
            .sport_reply = htons(120),
        /*  Destination L4 port in reply direction. Source port or routed packet
            will be replaced by this value. */
        .dport_reply = htons(110),
        /*  Protocol ID: UDP */
        .protocol = 17,
        /*  Flags: Do not open reply connection */
        .flags = htons(CTCMD_FLAGS_REP_DISABLED),
        /*  Associated route (456=emac1). This route will be used to forward
            packets matching this tracked connection (SIP+DIP+SPORT+DPORT+PROTO).
            Route must exist. To create a route see the FPP_CMD_IP_ROUTE. */
        .route_id = htonl(456),
    };
    fpp_ct_cmd_t ct2 =
    {
        .action = FPP_ACTION_REGISTER,
        /*  Source IP address: 120.41.48.100 */
        .saddr = htonl(0x78293064),
        /*  Destination IP address: 110.41.48.100 */
        .daddr = htonl(0x6e293064),
        .sport = htons(120),
        .dport = htons(110),
        .saddr_reply = htonl(0x0b293064),
        .daddr_reply = htonl(0x0c293064),
        .sport_reply = htons(11),
        .dport_reply = htons(12),
        .protocol = 17,
        .flags = htons(CTCMD_FLAGS_REP_DISABLED),
        .route_id = htonl(123)
    };
    /*  Register connection "ct1" */
    ret = fci_write(cl, FPP_CMD_IPV4_CONNTRACK, sizeof(ct1), (void *)&ct1);
    if (0 != ret)
    {
        printf("FPP_CMD_IPV4_CONNTRACK[FPP_ACTION_REGISTER] failed: %d\n", ret);
        return;
    }
    /*  Register connection "ct2" */
    ret = fci_write(cl, FPP_CMD_IPV4_CONNTRACK, sizeof(ct2), (void *)&ct2);
    if (0 != ret)
    {
        printf("FPP_CMD_IPV4_CONNTRACK[FPP_ACTION_REGISTER] failed: %d\n", ret);
        return;
    }
}
/*
 * @brief       Register bi-directional UPD connection to be fast-forwarded
 * @details     Add 2 routing table entries with a single conntrack. Traffic
 *              matching respective conntrack will be forwarded via physical
 *              interface given by matching route defined for both, the
 *              original as well as reply direction.
 *
 *              In case of no hit, packet will be sent to default logical
 *              interface (to host). Host can configure slow-path routing using
 *              standard OS-provided mechanisms to route rest of traffic (e.g.
 *              ICMP).
 * @param[in]   cl The FCI client instance
 */
void fci_router_register_bd_ipv4_conntrack(FCI_CLIENT *cl)
{
    int ret;
    fpp_ct_cmd_t ct1 =
    {
        /*  New connection */
        .action = FPP_ACTION_REGISTER,
        /*  Source IP address: 11.41.48.100 */
        .saddr = htonl(0x0b293064),
        /*  Destination IP address: 12.41.48.100 */
        .daddr = htonl(0x0c293064),
        /*  Source L4 port */
        .sport = htons(11),
        /*  Destination L4 port */
        .dport = htons(12),
```

```
        /*  Source IP address in reply direction. Equal to 'daddr' to
            disable NAT */
        .saddr_reply = htonl(0x0c293064),
        /*  Destination IP address in reply direction. Same as 'saddr' to
            disable NAT. */
        .daddr_reply = htonl(0x0b293064),
        /*  Source L4 port in reply direction. Equal to 'dport' to disable
            replacement. */
        .sport_reply = htons(12),
        /*  Destination L4 port in reply direction. Equal to 'sport' to disable
            replacement.*/
        .dport_reply = htons(11),
        /*  Protocol ID: UDP */
        .protocol = 17,
        /*  Flags: None. Create bi-directional connection. */
        .flags = htons(0),
        /*  Associated route (123=emac0, 456=emac1). This routes will be used to
            forward packets matching this tracked connection either in original
            or opposite, reply direction. Routes must exist. To create a route
            see the FPP_CMD_IP_ROUTE. */
        .route_id = htonl(456),
        .route_id_reply = htonl(123)
    };
    /*  Register connection "ct1" */
    ret = fci_write(cl, FPP_CMD_IPV4_CONNTRACK, sizeof(ct1), (void *)&ct1);
    if (0 != ret)
    {
        printf("FPP_CMD_IPV4_CONNTRACK[FPP_ACTION_REGISTER] failed: %d\n", ret);
        return;
    }
}
/*
 * @brief       Configure IPv4 router
 * @details     Create routes and conntracks to fast-forward traffic between
 *              EMAC0 and EMAC1. Put both EMACs to Router mode and enable them.
 * @param[in]   cl The FCI client instance
 */
void fci_setup_ipv4_router(FCI_CLIENT *cl)
{
    /*  Reset the router */
    fci_router_reset(cl);
    /*  Create routes */
    fci_router_register_ipv4_routes(cl);
    /*  Create conntracks */
    fci_router_register_ipv4_conntracks(cl);
    /*  Set interface mode */
    fci_phy_if_set_mode(cl, "emac0", FPP_IF_OP_ROUTER);
    fci_phy_if_set_mode(cl, "emac1", FPP_IF_OP_ROUTER);
    /*  Enable interfaces */
    fci_phy_if_enable(cl, "emac0");
    fci_phy_if_enable(cl, "emac1");
    /*
        Now traffic received via EMAC0 or EMAC1 and matching conntracks will be
        routed via interfaces defined by routes.
    */
}
/*
 * @brief       Configure IPv4 router with NAT
 * @details     Create routes and conntracks to fast-forward traffic between
 *              EMAC0 and EMAC1 and modify IP addresses and port numbers. Put
 *              both EMACs to Router mode and enable them.
 * @param[in]   cl The FCI client instance
 */
void fci_setup_ipv4_router_nat(FCI_CLIENT *cl)
{
    /*  Reset the router */
    fci_router_reset(cl);
    /*  Create routes */
    fci_router_register_ipv4_routes(cl);
    /*  Create conntracks with NAT enabled */
    fci_router_register_ipv4_conntracks_nat(cl);
    /*  Set interface mode */
    fci_phy_if_set_mode(cl, "emac0", FPP_IF_OP_ROUTER);
```

```
    fci_phy_if_set_mode(cl, "emac1", FPP_IF_OP_ROUTER);
    /*  Enable interfaces */
    fci_phy_if_enable(cl, "emac0");
    fci_phy_if_enable(cl, "emac1");
    /*
        Now traffic received via EMAC0 or EMAC1 and matching conntracks will be
        routed via interfaces defined by routes. Each routed packet will be
        modified in way that its source and destination IP address and source
        and destination port numbers will be replaced using configured values.
    */
}
/*
 * @brief       Configure IPv4 router using bi-directional conntrack
 * @details     Create routes and conntrack to fast-forward traffic between
 *              EMAC0 and EMAC1. Put both EMACs to Router mode and enable them.
 * @param[in]   cl The FCI client instance
 */
void fci_setup_ipv4_router_bd(FCI_CLIENT *cl)
{
    /*  Reset the router */
    fci_router_reset(cl);
    /*  Create routes */
    fci_router_register_ipv4_routes(cl);
    /*  Create bi-directional conntrack */
    fci_router_register_bd_ipv4_conntrack(cl);
    /*  Set interface mode */
    fci_phy_if_set_mode(cl, "emac0", FPP_IF_OP_ROUTER);
    fci_phy_if_set_mode(cl, "emac1", FPP_IF_OP_ROUTER);
    /*  Enable interfaces */
    fci_phy_if_enable(cl, "emac0");
    fci_phy_if_enable(cl, "emac1");
    /*
        Now traffic received via EMAC0 or EMAC1 and matching conntrack in both
        directions will be routed via interfaces defined by routes.
    */
}
```

8.3 fpp_cmd_ipv6_conntrack.c

```
/* =========================================================================
 *  Copyright 2020 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */
#include <stdio.h>
#include <stdlib.h>
```

```
#include <arpa/inet.h>
#include <errno.h>
#include "libfci.h"
#include "fpp.h"
#include "fpp_ext.h"
#include "fci_examples.h"
/*
 * @brief       Register two UPD connections to be fast-forwarded
 * @details     Add 2 routing table entries (conntracks). Traffic matching
 *              respective conntrack will be forwarded via physical interface
 *              given by matching route (fpp_rt_cmd_t.output_device) while its
 *              source MAC address will be replaced by MAC address of the output
 *              interface and destination MAC address will be replaced by the
 *              one defined by route (fpp_rt_cmd_t.dst_mac).
 *
 *              In case of no hit, packet will be sent to default logical
 *              interface (to host). Host can configure slow-path routing using
 *              standard OS-provided mechanisms to route rest of traffic (e.g.
 *              ICMP).
 * @param[in]   cl The FCI client instance
 */
void fci_router_register_ipv6_conntracks(FCI_CLIENT *cl)
{
    int ret;
    fpp_ct6_cmd_t ct1 =
    {
        /*  New connection */
        .action = FPP_ACTION_REGISTER,
        /*  Source IP address: ::aaaa */
        .saddr[0] = htonl(0),
        .saddr[1] = htonl(0),
        .saddr[2] = htonl(0),
        .saddr[3] = htonl(0x0000aaaa),
        /*  Destination IP address: ::bbbb */
        .daddr[0] = htonl(0),
        .daddr[1] = htonl(0),
        .daddr[2] = htonl(0),
        .daddr[3] = htonl(0x0000bbbb),
        /*  Source L4 port */
        .sport = htons(10),
        /*  Destination L4 port */
        .dport = htons(11),
        /*  Source IP address in reply direction. Equal to 'daddr' to disable
            replacement. */
        .saddr_reply[0] = htonl(0),
        .saddr_reply[1] = htonl(0),
        .saddr_reply[2] = htonl(0),
        .saddr_reply[3] = htonl(0x0000bbbb),
        /*  Destination IP address in reply direction. Same as 'saddr' to
            disable replacement. */
        .daddr_reply[0] = htonl(0),
        .daddr_reply[1] = htonl(0),
        .daddr_reply[2] = htonl(0),
        .daddr_reply[3] = htonl(0x0000aaaa),
        /*  Source L4 port in reply direction. Equal to 'dport' to disable
            replacement. */
        .sport_reply = htons(11),
        /*  Destination L4 port in reply direction. Equal to 'sport' to disable
            replacement.*/
        .dport_reply = htons(10),
        /*  Protocol ID: UDP */
        .protocol = 17,
        /*  Flags: Do not open reply connection */
        .flags = htons(CTCMD_FLAGS_REP_DISABLED),
        /*  Associated route (222=emac1). This route will be used to forward
            packets matching this tracked connection (SIP+DIP+SPORT+DPORT+PROTO).
            Route must exist. To create a route see the FPP_CMD_IP_ROUTE. */
        .route_id = htonl(222),
    };
    fpp_ct6_cmd_t ct2 =
    {
        .action = FPP_ACTION_REGISTER,
        /*  Source IP address: ::bbbb */
```

```
        .saddr[0] = htonl(0),
        .saddr[1] = htonl(0),
        .saddr[2] = htonl(0),
        .saddr[3] = htonl(0x0000bbbb),
        /* Destination IP address: ::aaaa */
        .daddr[0] = htonl(0),
        .daddr[1] = htonl(0),
        .daddr[2] = htonl(0),
        .daddr[3] = htonl(0x0000aaaa),
        .sport = htons(11),
        .dport = htons(10),
        .saddr_reply[0] = htonl(0),
        .saddr_reply[1] = htonl(0),
        .saddr_reply[2] = htonl(0),
        .saddr_reply[3] = htonl(0x0000aaaa),
        .daddr_reply[0] = htonl(0),
        .daddr_reply[1] = htonl(0),
        .daddr_reply[2] = htonl(0),
        .daddr_reply[3] = htonl(0x0000bbbb),
        .sport_reply = htons(10),
        .dport_reply = htons(11),
        .protocol = 17,
        .flags = htons(CTCMD_FLAGS_REP_DISABLED),
        .route_id = htonl(111)
    };
    /* Register connection "ct1" */
    ret = fci_write(cl, FPP_CMD_IPV6_CONNTRACK, sizeof(ct1), (void *)&ct1);
    if (0 != ret)
    {
        printf("FPP_CMD_IPV6_CONNTRACK[FPP_ACTION_REGISTER] failed: %d\n", ret);
        return;
    }
    /* Register connection "ct2" */
    ret = fci_write(cl, FPP_CMD_IPV6_CONNTRACK, sizeof(ct2), (void *)&ct2);
    if (0 != ret)
    {
        printf("FPP_CMD_IPV6_CONNTRACK[FPP_ACTION_REGISTER] failed: %d\n", ret);
        return;
    }
}
/*
 * @brief       Register bi-directional UPD connection to be fast-forwarded
 * @details     Add 2 routing table entries with a single conntrack. Traffic
 *              matching respective conntrack will be forwarded via physical
 *              interface given by matching route defined for both, the
 *              original as well as reply direction. Traffic matching
 *              respective direction will be forwarded via physical interface
 *              given by matching route (fpp_rt_cmd_t.output_device) while its
 *              source MAC address will be replaced by MAC address of the output
 *              interface and destination MAC address will be replaced by the
 *              one defined by route (fpp_rt_cmd_t.dst_mac).
 *
 *              In case of no hit, packet will be sent to default logical
 *              interface (to host). Host can configure slow-path routing using
 *              standard OS-provided mechanisms to route rest of traffic (e.g.
 *              ICMP).
 * @param[in]   cl The FCI client instance
 */
void fci_router_register_bd_ipv6_conntrack(FCI_CLIENT *cl)
{
    int ret;
    fpp_ct6_cmd_t ct1 =
    {
        /* New connection */
        .action = FPP_ACTION_REGISTER,
        /* Source IP address: ::aaaa */
        .saddr[0] = htonl(0),
        .saddr[1] = htonl(0),
        .saddr[2] = htonl(0),
        .saddr[3] = htonl(0x0000aaaa),
        /* Destination IP address: ::bbbb */
        .daddr[0] = htonl(0),
        .daddr[1] = htonl(0),
```

```
        .daddr[2] = htonl(0),
        .daddr[3] = htonl(0x0000bbbb),
        /*  Source L4 port */
        .sport = htons(10),
        /*  Destination L4 port */
        .dport = htons(11),
        /*  Source IP address in reply direction. Same as 'saddr' to
            disable replacement. */
        .saddr_reply[0] = htonl(0),
        .saddr_reply[1] = htonl(0),
        .saddr_reply[2] = htonl(0),
        .saddr_reply[3] = htonl(0x0000bbbb),
        /*  Destination IP address in reply direction. Same as 'saddr' to
            disable replacement. */
        .daddr_reply[0] = htonl(0),
        .daddr_reply[1] = htonl(0),
        .daddr_reply[2] = htonl(0),
        .daddr_reply[3] = htonl(0x0000aaaa),
        /*  Source L4 port in reply direction. Equal to 'dport' to disable
            replacement. */
        .sport_reply = htons(11),
        /*  Destination L4 port in reply direction. Equal to 'dport' to disable
            replacement. */
        .dport_reply = htons(10),
        /*  Protocol ID: UDP */
        .protocol = 17,
        /*  Flags: Create connection also in reply direction */
        .flags = htons(0),
        /*  Associated route (111=emac0, 222=emac1). This route will be used to forward
            packets matching this tracked connection (SIP+DIP+SPORT+DPORT+PROTO).
            Route must exist. To create a route see the FPP_CMD_IP_ROUTE. */
        .route_id = htonl(222),
        .route_id_reply = htonl(111),
    };
    /*  Register connection "ct1" */
    ret = fci_write(cl, FPP_CMD_IPV6_CONNTRACK, sizeof(ct1), (void *)&ct1);
    if (0 != ret)
    {
        printf("FPP_CMD_IPV6_CONNTRACK[FPP_ACTION_REGISTER] failed: %d\n", ret);
        return;
    }
}
/*
 * @brief       Configure IPv6 router
 * @details     Create routes and conntracks to fast-forward traffic between
 *              EMAC0 and EMAC1. Put both EMACs to Router mode and enable them.
 * @param[in]   cl The FCI client instance
 */
void fci_setup_ipv6_router(FCI_CLIENT *cl)
{
    /*  Reset the router */
    fci_router_reset(cl);
    /*  Create routes */
    fci_router_register_ipv6_routes(cl);
    /*  Create conntracks */
    fci_router_register_ipv6_conntracks(cl);
    /*  Set interface mode */
    fci_phy_if_set_mode(cl, "emac0", FPP_IF_OP_ROUTER);
    fci_phy_if_set_mode(cl, "emac1", FPP_IF_OP_ROUTER);
    /*  Enable interfaces */
    fci_phy_if_enable(cl, "emac0");
    fci_phy_if_enable(cl, "emac1");
    /*
        Now traffic received via EMAC0 or EMAC1 and matching conntracks will be
        routed via interfaces defined by routes.
    */
}
/*
 * @brief       Configure IPv6 router using bi-directional conntrack
 * @details     Create routes and conntrack to fast-forward traffic between
 *              EMAC0 and EMAC1 and modify IP addresses and port numbers. Put
 *              both EMACs to Router mode and enable them.
 * @param[in]   cl The FCI client instance
```

```
 */
void fci_setup_ipv6_router_bd(FCI_CLIENT *cl)
{
    /* Reset the router */
    fci_router_reset(cl);
    /* Create routes */
    fci_router_register_ipv6_routes(cl);
    /* Create bi-directional conntrack */
    fci_router_register_bd_ipv6_conntrack(cl);
    /* Set interface mode */
    fci_phy_if_set_mode(cl, "emac0", FPP_IF_OP_ROUTER);
    fci_phy_if_set_mode(cl, "emac1", FPP_IF_OP_ROUTER);
    /* Enable interfaces */
    fci_phy_if_enable(cl, "emac0");
    fci_phy_if_enable(cl, "emac1");
    /*
        Now traffic received via EMAC0 or EMAC1 and matching conntracks will be
        routed via interfaces defined by routes. Each routed packet will be
        modified in way that its source and destination IP address and source
        and destination port numbers will be replaced using configured values.
    */
}
```

8.4 fpp_cmd_log_if.c

```
/* =========================================================================
 *  Copyright 2020 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <errno.h>
#include <string.h>
#include "libfci.h"
#include "fpp.h"
#include "fpp_ext.h"
#include "fci_examples.h"
/*
 *  @brief      Print all logical interfaces
 *  @param[in]  cl The FCI client instance
 */
void fci_log_if_print_all(FCI_CLIENT *cl)
{
    fpp_log_if_cmd_t rep, cmd;
```

```
    unsigned short replen;
    int ret;
    /*  Get exclusive access to interfaces */
    if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_IF_LOCK_SESSION, 0, NULL)))
    {
        printf("FPP_CMD_IF_LOCK_SESSION failed: %d\n", ret);
    }
    else
    {
        /*  Get all interfaces */
        cmd.action = FPP_ACTION_QUERY;
        ret = fci_query(cl, FPP_CMD_LOG_IF, sizeof(cmd), (unsigned short *)&cmd,
                        &replen, (unsigned short *)&rep);
        while (FPP_ERR_OK == ret)
        {
            printf("%02d %s (%-5s): Flags: 0x%04x, Egress: 0x%08x, MatchRules: 0x%08x\n",
                        ntohl(rep.id), rep.name, rep.parent_name, rep.flags,
                                ntohl(rep.egress), ntohl(rep.match));
            cmd.action = FPP_ACTION_QUERY_CONT;
            ret = fci_query(cl, FPP_CMD_LOG_IF, sizeof(cmd), (unsigned short *)&cmd,
                        &replen, (unsigned short *)&rep);
        }
        /*  Unlock interfaces */
        if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_IF_UNLOCK_SESSION, 0, NULL)))
        {
            printf("FPP_CMD_IF_UNLOCK_SESSION failed: %d\n", ret);
        }
    }
}
/*
 * @brief       Create IPC channel
 * @details     Add logical interface on hif0 to send certain traffic to hif2. This allows
 *              setup of IPC channel between two host cores. Packets transmitted by host CPU
 *              via HIF channel 0 will be classified and in case that they contain destination
 *              IP address equal to 14.41.48.1 OR VLAN tag equal to 123, they will be
 *              forwarded to HIF channel 2, otherwise they will follow the default path.
 *
 *              In case the host sitting on hif2 would require similar packet distribution
 *              to hif0, another logical interface needs to be created on hif2.
 *
 *              This example utilizes the Flexible Router operation mode of hif0.
 * @param[in]   cl The FCI client instance
 */
void fci_add_ipc_log_if(FCI_CLIENT *cl)
{
    fpp_log_if_cmd_t cmd = {0};
    int ret;
    int dst_id = fci_phy_if_get_id(cl, "hif2");
    if (dst_id < 0)
    {
        printf("Could not get destination interface ID\n");
        return;
    }
    /*  Get exclusive access to interfaces */
    if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_IF_LOCK_SESSION, 0, NULL)))
    {
        printf("FPP_CMD_IF_LOCK_SESSION failed: %d\n", ret);
    }
    else
    {
        /*  Add logical interface to 'src' physical interface */
        cmd.action = FPP_ACTION_REGISTER;
        strncpy(cmd.name, "ipc0", sizeof(cmd.name)-1);
        strncpy(cmd.parent_name, "hif0", sizeof(cmd.parent_name)-1);
        ret = fci_write(cl, FPP_CMD_LOG_IF, sizeof(cmd), (unsigned short *)&cmd);
        if (FPP_ERR_OK != ret)
        {
            printf("ipc0 could not be created: %d\n", ret);
        }
        else
        {
            /*  Configure the new interface */
            cmd.action = FPP_ACTION_UPDATE;
```

```
            cmd.match = htonl(FPP_IF_MATCH_DIP|FPP_IF_MATCH_VLAN);
            cmd.arguments.v4.dip = htonl(0x0e293001); /* 14.41.48.1 */
            cmd.arguments.vlan = htons(123);
            cmd.flags = FPP_IF_ENABLED|FPP_IF_MATCH_OR;
            cmd.egress = htonl(1 « dst_id);
            ret = fci_write(cl, FPP_CMD_LOG_IF, sizeof(cmd), (unsigned short *)&cmd);
            if (FPP_ERR_OK != ret)
            {
                printf("Can't update: %d\n", ret);
            }
        }
        /*  Unlock interfaces */
        if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_IF_UNLOCK_SESSION, 0, NULL)))
        {
            printf("FPP_CMD_IF_UNLOCK_SESSION failed: %d\n", ret);
        }
    }
    /*  Configure pfe0 to start using logical interface-based
        classification of ingress traffic (Flexible Router) */
    fci_phy_if_set_mode(cl, "hif0", FPP_IF_OP_FLEXIBLE_ROUTER);

    /*  Enable the hif0 and hif2 */
    fci_phy_if_enable(cl, "hif0");
    fci_phy_if_enable(cl, "hif2");
}
/*
 * @brief       Delete a logical interface
 * @param[in]   cl The FCI client instance
 * @param[in]   name Name of the logical interface to remove
 */
void fci_log_if_del(FCI_CLIENT *cl, char *name)
{
    fpp_log_if_cmd_t cmd = {0};
    int ret;
    /*  Get exclusive access to interfaces */
    if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_IF_LOCK_SESSION, 0, NULL)))
    {
        printf("FPP_CMD_IF_LOCK_SESSION failed: %d\n", ret);
    }
    else
    {
        /*  Remove logical interface */
        cmd.action = FPP_ACTION_DEREGISTER;
        strncpy(cmd.name, name, sizeof(cmd.name)-1);
        ret = fci_write(cl, FPP_CMD_LOG_IF, sizeof(cmd), (unsigned short *)&cmd);
        if (FPP_ERR_OK != ret)
        {
            printf("ipc0 could not be deleted: %d\n", ret);
        }
        /*  Unlock interfaces */
        if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_IF_UNLOCK_SESSION, 0, NULL)))
        {
            printf("FPP_CMD_IF_UNLOCK_SESSION failed: %d\n", ret);
        }
    }
}
```

8.5 fpp_cmd_phy_if.c

```
/* ========================================================================
 *  Copyright 2020 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
```

```
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <errno.h>
#include <string.h>
#include "libfci.h"
#include "fpp.h"
#include "fpp_ext.h"
#include "fci_examples.h"
/*
 * @brief       Get QUERY response by interface name
 * @param[in]   cl The FCI client instance
 * @param[in]   name Physical interface name
 * @param[out]  phy_if Pointer where the response shall be written
 * @return      1 if success, zero otherwise
 */
int fci_phy_if_get_by_name(FCI_CLIENT *cl, char *name, fpp_phy_if_cmd_t *phy_if)
{
    fpp_phy_if_cmd_t rep = {0}, cmd = {0};
    unsigned short replen;
    int ret, retval = 0;
    /*  Get exclusive access to interfaces */
    if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_IF_LOCK_SESSION, 0, NULL)))
    {
        printf("FPP_CMD_IF_LOCK_SESSION failed: %d\n", ret);
    }
    else
    {
        /*  Get all interfaces */
        cmd.action = FPP_ACTION_QUERY;
        ret = fci_query(cl, FPP_CMD_PHY_IF, sizeof(cmd), (unsigned short *)&cmd,
                        &replen, (unsigned short *)&rep);
        while (FPP_ERR_OK == ret)
        {
            if (0 == strcmp(name, rep.name))
            {
                memcpy(phy_if, &rep, sizeof(*phy_if));
                retval = 1;
                break;
            }
            else
            {
                cmd.action = FPP_ACTION_QUERY_CONT;
                ret = fci_query(cl, FPP_CMD_PHY_IF, sizeof(cmd), (unsigned short *)&cmd,
                                &replen, (unsigned short *)&rep);
            }
        }
        /*  Unlock interfaces */
        if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_IF_UNLOCK_SESSION, 0, NULL)))
        {
            printf("FPP_CMD_IF_UNLOCK_SESSION failed: %d\n", ret);
        }
    }
    return retval;
}
```

```
/*
 * @brief       Enable physical interface
 * @param[in]   cl The FCI client instance
 * @param[in]   name Physical interface name
 */
void fci_phy_if_enable(FCI_CLIENT *cl, char *name)
{
    fpp_phy_if_cmd_t cmd;
    int ret;
    if (fci_phy_if_get_by_name(cl, name, &cmd))
    {
        /*  Get exclusive access to interfaces */
        if (FPP_ERR_OK != fci_write(cl, FPP_CMD_IF_LOCK_SESSION, 0, NULL))
        {
            printf("FPP_CMD_IF_LOCK_SESSION failed\n");
        }
        else
        {
            cmd.action = FPP_ACTION_UPDATE;
            cmd.flags |= FPP_IF_ENABLED;
            if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_PHY_IF, sizeof(cmd),
                                        (unsigned short *)&cmd)))
            {
                printf("%s enable failed: %d\n", name, ret);
            }
        }
        /*  Unlock interfaces */
        if (FPP_ERR_OK != fci_write(cl, FPP_CMD_IF_UNLOCK_SESSION, 0, NULL))
        {
            printf("FPP_CMD_IF_UNLOCK_SESSION failed\n");
        }
    }
    else
    {
        printf("%s not found\n", name);
    }
}
/*
 * @brief       Disable physical interface
 * @param[in]   cl The FCI client instance
 * @param[in]   name Physical interface name
 */
void fci_phy_if_disable(FCI_CLIENT *cl, char *name)
{
    fpp_phy_if_cmd_t cmd;
    int ret;
    if (fci_phy_if_get_by_name(cl, name, &cmd))
    {
        cmd.action = FPP_ACTION_UPDATE;
        cmd.flags &= ~FPP_IF_ENABLED;
        /*  Get exclusive access to interfaces */
        if (FPP_ERR_OK != fci_write(cl, FPP_CMD_IF_LOCK_SESSION, 0, NULL))
        {
            printf("FPP_CMD_IF_LOCK_SESSION failed\n");
        }
        else
        {
            if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_PHY_IF, sizeof(cmd),
                                            (unsigned short *)&cmd)))
            {
                printf("%s disable failed: %d\n", name, ret);
            }
        }
        /*  Unlock interfaces */
        if (FPP_ERR_OK != fci_write(cl, FPP_CMD_IF_UNLOCK_SESSION, 0, NULL))
        {
            printf("FPP_CMD_IF_UNLOCK_SESSION failed\n");
        }
    }
    else
    {
        printf("%s not found\n", name);
    }
```

```
}
/*
 * @brief        Print all physical interfaces
 * @param[in]    The FCI client instance
 */
void fci_phy_if_print_all(FCI_CLIENT *cl)
{
    fpp_phy_if_cmd_t rep = {0}, cmd = {0};
    unsigned short replen;
    int ret;
    /*  Get exclusive access to interfaces */
    if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_IF_LOCK_SESSION, 0, NULL)))
    {
        printf("FPP_CMD_IF_LOCK_SESSION failed: %d\n", ret);
    }
    else
    {
        /*  Get all interfaces */
        cmd.action = FPP_ACTION_QUERY;
        ret = fci_query(cl, FPP_CMD_PHY_IF, sizeof(cmd), (unsigned short *)&cmd,
                    &replen, (unsigned short *)&rep);
        while (FPP_ERR_OK == ret)
        {
            printf("%02d %-5s: Mode: 0x%x, Flags: 0x%04x\n",
                        ntohl(rep.id), rep.name, rep.mode, rep.flags);
            cmd.action = FPP_ACTION_QUERY_CONT;
            ret = fci_query(cl, FPP_CMD_PHY_IF, sizeof(cmd), (unsigned short *)&cmd,
                        &replen, (unsigned short *)&rep);
        }
    }
    /*  Unlock interfaces */
    if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_IF_UNLOCK_SESSION, 0, NULL)))
    {
        printf("FPP_CMD_IF_UNLOCK_SESSION failed: %d\n", ret);
    }
}
/*
 * @brief        Get physical interface ID by name
 * @param[in]    The FCI client instance
 * @param[in]    name Name of physical interface
 * @return       Physical interface ID (in host byte order) or -1 if failed.
 */
int fci_phy_if_get_id(FCI_CLIENT *cl, char *name)
{
    fpp_phy_if_cmd_t rep = {0};
    int ret;
    /*  Get reply data by name */
    if (fci_phy_if_get_by_name(cl, name, &rep))
    {
        ret = ntohl(rep.id);
    }
    else
    {
        ret = -1;
    }
    return ret;
}
/*
 * @brief        Change mode and enable physical interface
 * @param[in]    cl The FCI client instance
 * @param[in]    id Physical interface ID
 * @param[in]    mode New operation mode to be set
 */
void fci_phy_if_set_mode(FCI_CLIENT *cl, char *name, fpp_phy_if_op_mode_t mode)
{
    fpp_phy_if_cmd_t rep;
    int ret;
    /*  Get interface reply by name */
    if (fci_phy_if_get_by_name(cl, name, &rep))
    {
        /*  Get exclusive access to interfaces */
        if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_IF_LOCK_SESSION, 0, NULL)))
        {
```

```
            printf("FPP_CMD_IF_LOCK_SESSION failed: %d\n", ret);
        }
        else
        {
            /*  Change the mode */
            rep.action = FPP_ACTION_UPDATE;
            rep.mode = mode;
            if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_PHY_IF,
                                        sizeof(rep), (unsigned short *)&rep)))
            {
                printf("Mode change failed: %d\n", ret);
            }
            /*  Unlock interfaces */
            if (FPP_ERR_OK != (ret = fci_write(cl, FPP_CMD_IF_UNLOCK_SESSION, 0, NULL)))
            {
                printf("FPP_CMD_IF_UNLOCK_SESSION failed: %d\n", ret);
            }
        }
    }
    else
    {
        printf("%s not found\n", name);
    }
}
```

Index

accepted
    fpp_algo_stats_t, 52
action
    fpp_ct6_cmd_t, 54
    fpp_ct_cmd_t, 56
    fpp_fp_rule_cmd_t, 59
    fpp_fp_table_cmd_t, 60
    fpp_l2_bd_cmd_t, 64
    fpp_log_if_cmd_t, 66
    fpp_phy_if_cmd_t, 69
    fpp_rt_cmd_t, 72
arguments
    fpp_log_if_cmd_t, 68

block_state
    fpp_phy_if_cmd_t, 70
BS_BLOCKED
    LibFCI, 44
BS_FORWARD_ONLY
    LibFCI, 44
BS_LEARN_ONLY
    LibFCI, 44
BS_NORMAL
    LibFCI, 44

CTCMD_FLAGS_REP_DISABLED
    LibFCI, 41

daddr
    fpp_ct6_cmd_t, 54
    fpp_ct_cmd_t, 57
daddr_reply
    fpp_ct6_cmd_t, 55
    fpp_ct_cmd_t, 57
discarded
    fpp_algo_stats_t, 52
    fpp_phy_if_stats_t, 71
dmac
    fpp_if_m_args_t, 63
dport
    fpp_ct6_cmd_t, 54
    fpp_ct_cmd_t, 57
    fpp_if_m_args_t, 62
dport_reply
    fpp_ct6_cmd_t, 55
    fpp_ct_cmd_t, 58
dst_mac
    fpp_rt_cmd_t, 73

egress
    fpp_log_if_cmd_t, 67
    fpp_phy_if_stats_t, 71
ethtype
    fpp_if_m_args_t, 62

fci_catch
    LibFCI, 47
FCI_CB_CONTINUE
    LibFCI, 46
fci_cb_retval_t
    LibFCI, 45
FCI_CB_STOP
    LibFCI, 46
FCI_CFG_FORCE_LEGACY_API
    LibFCI, 41
FCI_CLIENT, 51
FCI_CLIENT_DEFAULT
    LibFCI, 45
fci_client_type_t
    LibFCI, 45
fci_close
    LibFCI, 46
fci_cmd
    LibFCI, 47
FCI_GROUP_CATCH
    LibFCI, 45
FCI_GROUP_NONE
    LibFCI, 45
fci_mcast_groups_t
    LibFCI, 45
fci_open
    LibFCI, 46
fci_query
    LibFCI, 48

fci_register_cb
LibFCI, 50
fci_write
LibFCI, 49
flags
fpp_ct6_cmd_t, 55
fpp_ct_cmd_t, 58
fpp_l2_bd_cmd_t, 65
fpp_log_if_cmd_t, 67
fpp_phy_if_cmd_t, 69
fpp_rt_cmd_t, 73
FP_ACCEPT
LibFCI, 44
FP_NEXT_RULE
LibFCI, 44
FP_OFFSET_FROM_L2_HEADER
LibFCI, 45
FP_OFFSET_FROM_L3_HEADER
LibFCI, 45
FP_OFFSET_FROM_L4_HEADER
LibFCI, 45
FP_REJECT
LibFCI, 44
fp_table0
fpp_if_m_args_t, 63
fp_table1
fpp_if_m_args_t, 63
fpp.h, 75
FPP_CMD_IP_ROUTE, 80
FPP_CMD_IPV4_CONNTRACK, 76
FPP_CMD_IPV4_RESET, 82
FPP_CMD_IPV4_SET_TIMEOUT, 82
FPP_CMD_IPV6_CONNTRACK, 78
FPP_CMD_IPV6_RESET, 82
fpp_algo_stats_t, 51
accepted, 52
discarded, 52
processed, 52
rejected, 52
fpp_buf_cmd_t, 52
len, 53
payload, 53
FPP_CMD_DATA_BUF_AVAIL
LibFCI, 40
FPP_CMD_DATA_BUF_PUT
LibFCI, 40
FPP_CMD_FP_FLEXIBLE_FILTER
LibFCI, 39
FPP_CMD_FP_RULE
LibFCI, 38
FPP_CMD_FP_TABLE
LibFCI, 36
FPP_CMD_IF_LOCK_SESSION
LibFCI, 32
FPP_CMD_IF_UNLOCK_SESSION
LibFCI, 32
FPP_CMD_IP_ROUTE
fpp.h, 80
FPP_CMD_IPV4_CONNTRACK
fpp.h, 76
FPP_CMD_IPV4_CONNTRACK_CHANGE
LibFCI, 41
FPP_CMD_IPV4_RESET
fpp.h, 82
FPP_CMD_IPV4_SET_TIMEOUT
fpp.h, 82
FPP_CMD_IPV6_CONNTRACK
fpp.h, 78
FPP_CMD_IPV6_CONNTRACK_CHANGE
LibFCI, 41
FPP_CMD_IPV6_RESET
fpp.h, 82
FPP_CMD_L2_BD
LibFCI, 33
FPP_CMD_LOG_IF
LibFCI, 30
FPP_CMD_PHY_IF
LibFCI, 29
fpp_ct6_cmd_t, 53
action, 54
daddr, 54
daddr_reply, 55
dport, 54
dport_reply, 55
flags, 55
protocol, 55
route_id, 55
route_id_reply, 55
saddr, 54
saddr_reply, 55
sport, 54
sport_reply, 55
fpp_ct_cmd_t, 56
action, 56
daddr, 57
daddr_reply, 57

dport, 57
dport_reply, 58
flags, 58
protocol, 58
route_id, 58
route_id_reply, 58
saddr, 57
saddr_reply, 57
sport, 57
sport_reply, 57
fpp_ext.h, 83
fpp_fp_offset_from_t
LibFCI, 44
fpp_fp_rule_cmd_t, 58
action, 59
r, 59
fpp_fp_rule_match_action_t
LibFCI, 44
fpp_fp_rule_props_t, 59
fpp_fp_table_cmd_t, 60
action, 60
position, 60
r, 60
rule_name, 60
table_name, 60
FPP_IF_DISCARD
LibFCI, 42
FPP_IF_ENABLED
LibFCI, 42
fpp_if_flags_t
LibFCI, 42
fpp_if_m_args_t, 61
dmac, 63
dport, 62
ethtype, 62
fp_table0, 63
fp_table1, 63
hif_cookie, 63
proto, 62
smac, 62
sport, 62
v4, 62
v6, 62
vlan, 61
fpp_if_m_rules_t
LibFCI, 42
FPP_IF_MATCH_DIP
LibFCI, 43
FPP_IF_MATCH_DIP6
LibFCI, 43
FPP_IF_MATCH_DMAC
LibFCI, 43
FPP_IF_MATCH_DPORT
LibFCI, 43
FPP_IF_MATCH_ETHTYPE
LibFCI, 43
FPP_IF_MATCH_FP0
LibFCI, 43
FPP_IF_MATCH_FP1
LibFCI, 43
FPP_IF_MATCH_HIF_COOKIE
LibFCI, 43
FPP_IF_MATCH_OR
LibFCI, 42
FPP_IF_MATCH_PROTO
LibFCI, 43
FPP_IF_MATCH_RESERVED7
LibFCI, 43
FPP_IF_MATCH_RESERVED8
LibFCI, 43
FPP_IF_MATCH_SIP
LibFCI, 43
FPP_IF_MATCH_SIP6
LibFCI, 43
FPP_IF_MATCH_SMAC
LibFCI, 43
FPP_IF_MATCH_SPORT
LibFCI, 43
FPP_IF_MATCH_TYPE_ARP
LibFCI, 43
FPP_IF_MATCH_TYPE_BCAST
LibFCI, 43
FPP_IF_MATCH_TYPE_ETH
LibFCI, 43
FPP_IF_MATCH_TYPE_ICMP
LibFCI, 43
FPP_IF_MATCH_TYPE_IGMP
LibFCI, 43
FPP_IF_MATCH_TYPE_IPV4
LibFCI, 43
FPP_IF_MATCH_TYPE_IPV6
LibFCI, 43
FPP_IF_MATCH_TYPE_IPX
LibFCI, 43
FPP_IF_MATCH_TYPE_MCAST
LibFCI, 43

FPP_IF_MATCH_TYPE_PPPOE
LibFCI, 43
FPP_IF_MATCH_TYPE_TCP
LibFCI, 43
FPP_IF_MATCH_TYPE_UDP
LibFCI, 43
FPP_IF_MATCH_TYPE_VLAN
LibFCI, 43
FPP_IF_MATCH_VLAN
LibFCI, 43
FPP_IF_MIRROR
LibFCI, 42
FPP_IF_OP_BRIDGE
LibFCI, 42
FPP_IF_OP_DEFAULT
LibFCI, 42
FPP_IF_OP_DISABLED
LibFCI, 42
FPP_IF_OP_FLEXIBLE_ROUTER
LibFCI, 42
FPP_IF_OP_ROUTER
LibFCI, 42
FPP_IF_OP_VLAN_BRIDGE
LibFCI, 42
FPP_IF_PROMISC
LibFCI, 42
fpp_l2_bd_cmd_t, 63
action, 64
flags, 65
if_list, 65
mcast_hit, 64
mcast_miss, 64
ucast_hit, 64
ucast_miss, 64
untag_if_list, 65
vlan, 64
fpp_l2_bd_flags_t
LibFCI, 44
FPP_L2BR_DOMAIN_DEFAULT
LibFCI, 44
FPP_L2BR_DOMAIN_FALLBACK
LibFCI, 44
fpp_log_if_cmd_t, 65
action, 66
arguments, 68
egress, 67
flags, 67
id, 66
match, 67
name, 66
parent_id, 67
parent_name, 67
stats, 68
fpp_phy_if_block_state_t
LibFCI, 43
fpp_phy_if_cmd_t, 68
action, 69
block_state, 70
flags, 69
id, 69
mac_addr, 70
mirror, 70
mode, 70
name, 69
stats, 70
fpp_phy_if_op_mode_t
LibFCI, 42
fpp_phy_if_stats_t, 71
discarded, 71
egress, 71
ingress, 71
malformed, 71
fpp_rt_cmd_t, 72
action, 72
dst_mac, 73
flags, 73
id, 73
output_device, 73
fpp_timeout_cmd_t, 74

hif_cookie
fpp_if_m_args_t, 63

id
fpp_log_if_cmd_t, 66
fpp_phy_if_cmd_t, 69
fpp_rt_cmd_t, 73
if_list
fpp_l2_bd_cmd_t, 65
ingress
fpp_phy_if_stats_t, 71

len
fpp_buf_cmd_t, 53
LibFCI, 15
BS_BLOCKED, 44
BS_FORWARD_ONLY, 44

BS_LEARN_ONLY, 44
BS_NORMAL, 44
CTCMD_FLAGS_REP_DISABLED, 41
fci_catch, 47
FCI_CB_CONTINUE, 46
fci_cb_retval_t, 45
FCI_CB_STOP, 46
FCI_CFG_FORCE_LEGACY_API, 41
FCI_CLIENT_DEFAULT, 45
fci_client_type_t, 45
fci_close, 46
fci_cmd, 47
FCI_GROUP_CATCH, 45
FCI_GROUP_NONE, 45
fci_mcast_groups_t, 45
fci_open, 46
fci_query, 48
fci_register_cb, 50
fci_write, 49
FP_ACCEPT, 44
FP_NEXT_RULE, 44
FP_OFFSET_FROM_L2_HEADER, 45
FP_OFFSET_FROM_L3_HEADER, 45
FP_OFFSET_FROM_L4_HEADER, 45
FP_REJECT, 44
FPP_CMD_DATA_BUF_AVAIL, 40
FPP_CMD_DATA_BUF_PUT, 40
FPP_CMD_FP_FLEXIBLE_FILTER, 39
FPP_CMD_FP_RULE, 38
FPP_CMD_FP_TABLE, 36
FPP_CMD_IF_LOCK_SESSION, 32
FPP_CMD_IF_UNLOCK_SESSION, 32
FPP_CMD_IPV4_CONNTRACK_CHANGE, 41
FPP_CMD_IPV6_CONNTRACK_CHANGE, 41
FPP_CMD_L2_BD, 33
FPP_CMD_LOG_IF, 30
FPP_CMD_PHY_IF, 29
fpp_fp_offset_from_t, 44
fpp_fp_rule_match_action_t, 44
FPP_IF_DISCARD, 42
FPP_IF_ENABLED, 42
fpp_if_flags_t, 42
fpp_if_m_rules_t, 42
FPP_IF_MATCH_DIP, 43
FPP_IF_MATCH_DIP6, 43
FPP_IF_MATCH_DMAC, 43
FPP_IF_MATCH_DPORT, 43
FPP_IF_MATCH_ETHTYPE, 43
FPP_IF_MATCH_FP0, 43
FPP_IF_MATCH_FP1, 43
FPP_IF_MATCH_HIF_COOKIE, 43
FPP_IF_MATCH_OR, 42
FPP_IF_MATCH_PROTO, 43
FPP_IF_MATCH_RESERVED7, 43
FPP_IF_MATCH_RESERVED8, 43
FPP_IF_MATCH_SIP, 43
FPP_IF_MATCH_SIP6, 43
FPP_IF_MATCH_SMAC, 43
FPP_IF_MATCH_SPORT, 43
FPP_IF_MATCH_TYPE_ARP, 43
FPP_IF_MATCH_TYPE_BCAST, 43
FPP_IF_MATCH_TYPE_ETH, 43
FPP_IF_MATCH_TYPE_ICMP, 43
FPP_IF_MATCH_TYPE_IGMP, 43
FPP_IF_MATCH_TYPE_IPV4, 43
FPP_IF_MATCH_TYPE_IPV6, 43
FPP_IF_MATCH_TYPE_IPX, 43
FPP_IF_MATCH_TYPE_MCAST, 43
FPP_IF_MATCH_TYPE_PPPOE, 43
FPP_IF_MATCH_TYPE_TCP, 43
FPP_IF_MATCH_TYPE_UDP, 43
FPP_IF_MATCH_TYPE_VLAN, 43
FPP_IF_MATCH_VLAN, 43
FPP_IF_MIRROR, 42
FPP_IF_OP_BRIDGE, 42
FPP_IF_OP_DEFAULT, 42
FPP_IF_OP_DISABLED, 42
FPP_IF_OP_FLEXIBLE_ROUTER, 42
FPP_IF_OP_ROUTER, 42
FPP_IF_OP_VLAN_BRIDGE, 42
FPP_IF_PROMISC, 42
fpp_l2_bd_flags_t, 44
FPP_L2BR_DOMAIN_DEFAULT, 44
FPP_L2BR_DOMAIN_FALLBACK, 44
fpp_phy_if_block_state_t, 43
fpp_phy_if_op_mode_t, 42

libfci.h, 85

mac_addr
    fpp_phy_if_cmd_t, 70
malformed
    fpp_phy_if_stats_t, 71
match
    fpp_log_if_cmd_t, 67
mcast_hit

fpp_l2_bd_cmd_t, 64
mcast_miss
fpp_l2_bd_cmd_t, 64
mirror
fpp_phy_if_cmd_t, 70
mode
fpp_phy_if_cmd_t, 70

name
fpp_log_if_cmd_t, 66
fpp_phy_if_cmd_t, 69

output_device
fpp_rt_cmd_t, 73

parent_id
fpp_log_if_cmd_t, 67
parent_name
fpp_log_if_cmd_t, 67
payload
fpp_buf_cmd_t, 53
position
fpp_fp_table_cmd_t, 60
processed
fpp_algo_stats_t, 52
proto
fpp_if_m_args_t, 62
protocol
fpp_ct6_cmd_t, 55
fpp_ct_cmd_t, 58

r
fpp_fp_rule_cmd_t, 59
fpp_fp_table_cmd_t, 60
rejected
fpp_algo_stats_t, 52
route_id
fpp_ct6_cmd_t, 55
fpp_ct_cmd_t, 58
route_id_reply
fpp_ct6_cmd_t, 55
fpp_ct_cmd_t, 58
rule_name
fpp_fp_table_cmd_t, 60

saddr
fpp_ct6_cmd_t, 54
fpp_ct_cmd_t, 57
saddr_reply
fpp_ct6_cmd_t, 55
fpp_ct_cmd_t, 57
smac
fpp_if_m_args_t, 62
sport
fpp_ct6_cmd_t, 54
fpp_ct_cmd_t, 57
fpp_if_m_args_t, 62
sport_reply
fpp_ct6_cmd_t, 55
fpp_ct_cmd_t, 57
stats
fpp_log_if_cmd_t, 68
fpp_phy_if_cmd_t, 70

table_name
fpp_fp_table_cmd_t, 60

ucast_hit
fpp_l2_bd_cmd_t, 64
ucast_miss
fpp_l2_bd_cmd_t, 64
untag_if_list
fpp_l2_bd_cmd_t, 65

v4
fpp_if_m_args_t, 62
v6
fpp_if_m_args_t, 62
vlan
fpp_if_m_args_t, 61
fpp_l2_bd_cmd_t, 64