Mellanox NPS-400 L23Qos Application

Rev 2.2

Software Version 18.0300.00

www.mellanox.com

NOTE:
THIS HARDWARE, SOFTWARE OR TEST SUITE PRODUCT ("PRODUCT(S)") AND ITS RELATED DOCUMENTATION ARE PROVIDED BY MELLANOX TECHNOLOGIES "AS-IS" WITH ALL FAULTS OF ANY KIND AND SOLELY FOR THE PURPOSE OF AIDING THE CUSTOMER IN TESTING APPLICATIONS THAT USE THE PRODUCTS IN DESIGNATED SOLUTIONS. THE CUSTOMER'S MANUFACTURING TEST ENVIRONMENT HAS NOT MET THE STANDARDS SET BY MELLANOX TECHNOLOGIES TO FULLY QUALIFY THE PRODUCT(S) AND/OR THE SYSTEM USING IT. THEREFORE, MELLANOX TECHNOLOGIES CANNOT AND DOES NOT GUARANTEE OR WARRANT THAT THE PRODUCTS WILL OPERATE WITH THE HIGHEST QUALITY. ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT ARE DISCLAIMED. IN NO EVENT SHALL MELLANOX BE LIABLE TO CUSTOMER OR ANY THIRD PARTIES FOR ANY DIRECT, INDIRECT, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES OF ANY KIND (INCLUDING, BUT NOT LIMITED TO, PAYMENT FOR 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 FROM THE USE OF THE PRODUCT(S) AND RELATED DOCUMENTATION EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Mellanox Technologies
350 Oakmead Parkway Suite 100
Sunnyvale, CA 94085
U.S.A.
www.mellanox.com
Tel: (408) 970-3400
Fax: (408) 970-3403

Table of Contents

List of Figures

Release Update History

Release	Date	Description
Rev 2.2	Aug. 30, 2016	Relates to EZdk version 18.0300.00. No changes to the document.
2.1_Open	Aug. 14, 2016	Relates to EZdk version 2.1a Open. Added Section 5.1, “Compiling the Application”, on page 22.
	June 28, 2016	Relates to EZdk version 2.1a Open.

About this Manual

This application note describes operation of a simple Layer 2 switching and Layer 3 routing sample application, named *L23Qos* (Layer 2 and Layer 3 Quality of Service) on the NPS-400 network processor.

Note: Note that frames traverse the TM but without any WRED, QoS or WTD functionality.

Audience

This document is intended for software developers who will be developing applications for the NPS-400 network processor.

1 Overview

This document describes how to use the L23Qos application.

Depending upon the type of incoming frame there are two possible scenarios:

- "Layer 2 Switching" – The L2 switching algorithm is applied to frames that are not destined to the router's MAC address. Various VLANs are assumed in the sample. Frames with an unknown destination address or a broadcast DA will be broadcast to all the ports in the VLAN. This algorithm is designed according to IEEE 802.1Q-1988.
- "Layer 3 Routing" – The L3 routing algorithm is applied to IP frames that are destined to the router's MAC address.

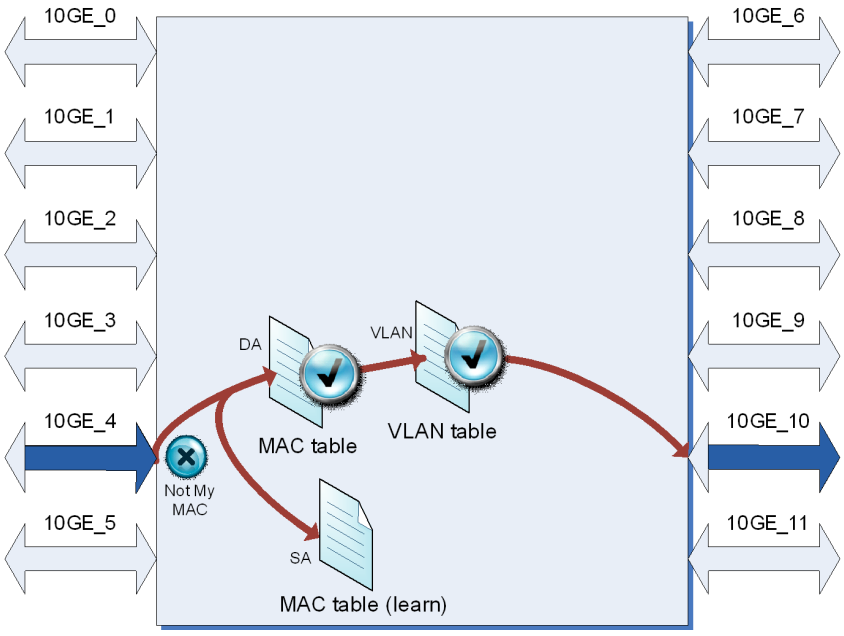
This sample handles Ethernet II frames with and without tags as well as IP frames at 600 Gbps. It implements MAC learning for all L2 frames. Checksum, TTL and IPv4 validation are performed for L3 frames.

The L23Qos application includes several sample frame files that may be used with the supplied projects (see "Supplied Files, Frames and Search Entries"). In addition, you may create your own frame files, according to the format described in the "Frame Formats and Search Entries" section, to match NPS-400 configurations other than the ones provided (such as different number of ports and different interfaces).

1.1 Layer 2 Switching

The L2 switching algorithm is applied to frames that are not destined to the router's MAC address. Various VLANs are assumed in the sample. For cases where the destination is known (Destination Address was matched in the MAC table ("DA Structure (hash)"), the frame is forwarded according to the output port obtained from the MAC table. The VLAN table ("VLAN Structure (table)") is also used as a validation that the input and output ports participate in the VLAN group.

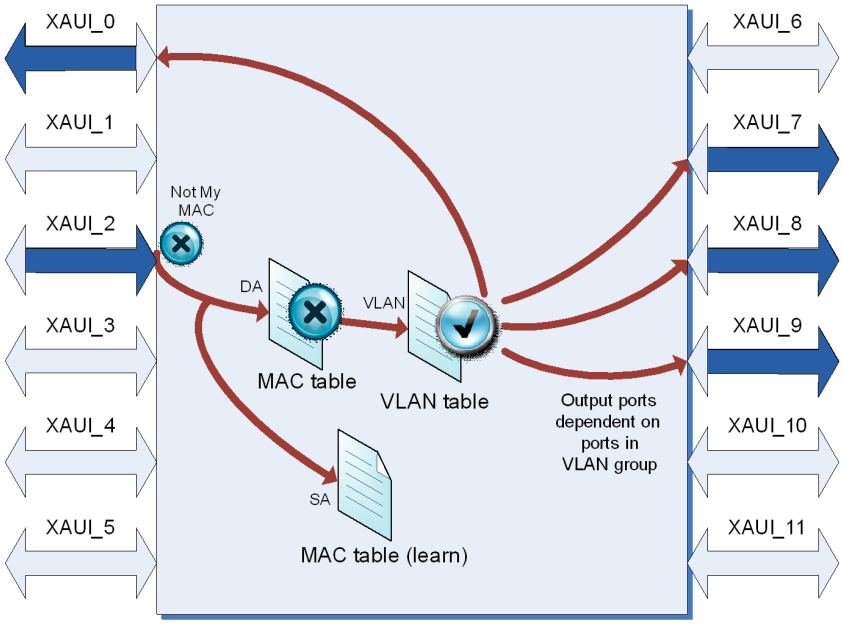
Figure 1: Layer 2 Switching for a Known Destination



Note: The behavior illustrated in this section is applicable when the default values are loaded to the search structures from the provided entries.txt file.

Frames with an unknown destination address or a broadcast DA will be broadcasted to all the ports in the VLAN group, excluding the input port. This algorithm is designed according to IEEE 802.1Q-1988. In this case, the "VLAN Structure (table)" provides all destination ports.

Figure 2: Layer 2 Switching for an Unknown Destination

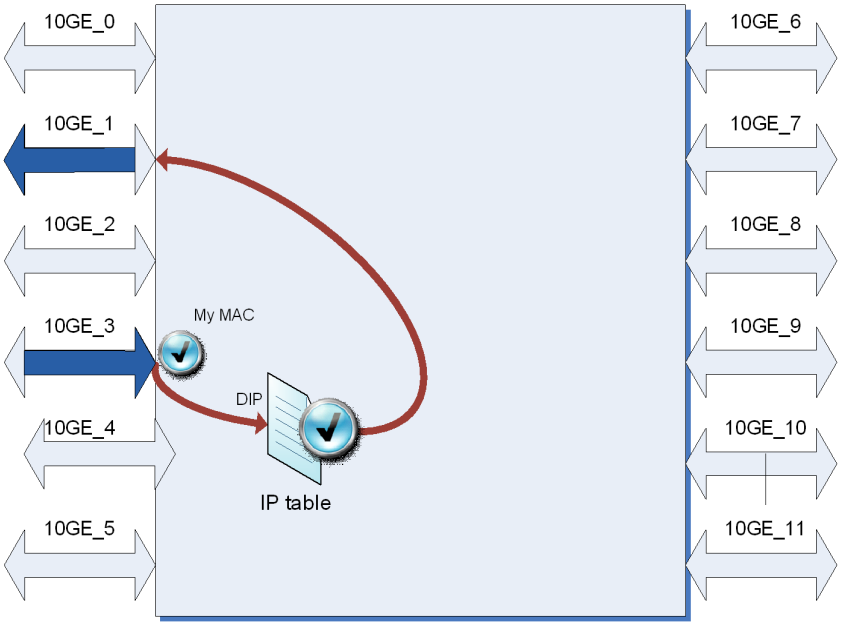


In cases of L2 switching, the packet content is not modified.

1.2 Layer 3 Routing

The L3 routing algorithm is applied to IP frames that are destined to the router's MAC address. A lookup of the Destination Address (DIP) is performed in the IP table ("DIP Structure") in order to determine the outgoing interface and frame modifications to be carried out.

Figure 3: Layer 3 Routing



In the L3 header, the TTL field of the IP header is decremented by one, and the checksum field is recalculated.

In the L2 header, the new DA and VLAN tags are replaced with values taken from the IP forwarding table ("DIP Structure"), whereas the new SA is taken from the output port configuration ("OUT_PORT Structure (table)").

2 Application Description

2.1 Data Flow

The main data flow of the application is as follows (see Figure 4,“L23Qos Flow Chart” on page 12 below):

Step 1. Receive a new frame and load current buffer data to CMEM.

Step 2. If an error occurred during data load, discard frame.

Step 3. If the frame’s logical id is loopback, broadcast the frame:

Step a. Select output port according to received port bitmap.

Step b. Send frame to TM according to QoS algorithm.

Step 4. Else

Step a. Check if frame is valid. If not, discard frame.

Step b. Decode the Ethernet header. If an error is found, discard the frame.

Step c. Lookup in “IN_PORT Structure (table)” with source port key. Results contain port’s MAC address.

Step d. If frame’s DA equals port’s MAC, handle as L3 Routing:

- Decode the IP header. If an error is found, discard the frame.
- Lookup in “DIP Structure” with destination IP address key.
- If DIP lookup didn’t match, use HOST_PORT (as defined in defs.h) as output port.

Step e. Lookup output port. If port invalid, discard frame.

Step f. If IP TTL equals 0, discard the frame.

Step g. If source port equals out port from DIP result, discard the frame.

Step h. If (add_out_tag) bit from output port results is on, add C-Vlan. (If C-Vlan already exists, only replace the VLAN tag.)

Step i. Decrement TTL, update SA to port’s MAC and DA according to DIP result.

Step j. Send frame to TM according to QoS algorithm description.

Step 5. Else (frame’s DA differs from port’s MAC), handle as L2 Switching:

Step a. Lookup in “VLAN Structure (table)”. (If no VLAN exists, use default VLAN from IN_PORT result.)

Step b. If no match in VLAN table, discard the frame.

Step c. Verify that the source port is part of the VLAN (port number corresponds to bit offset in the bitmap). If not, discard the frame.

Step d. Learn SA+Vlan. Lookup in “DA Structure (hash)”.

Step e. Based on DA lookup match:

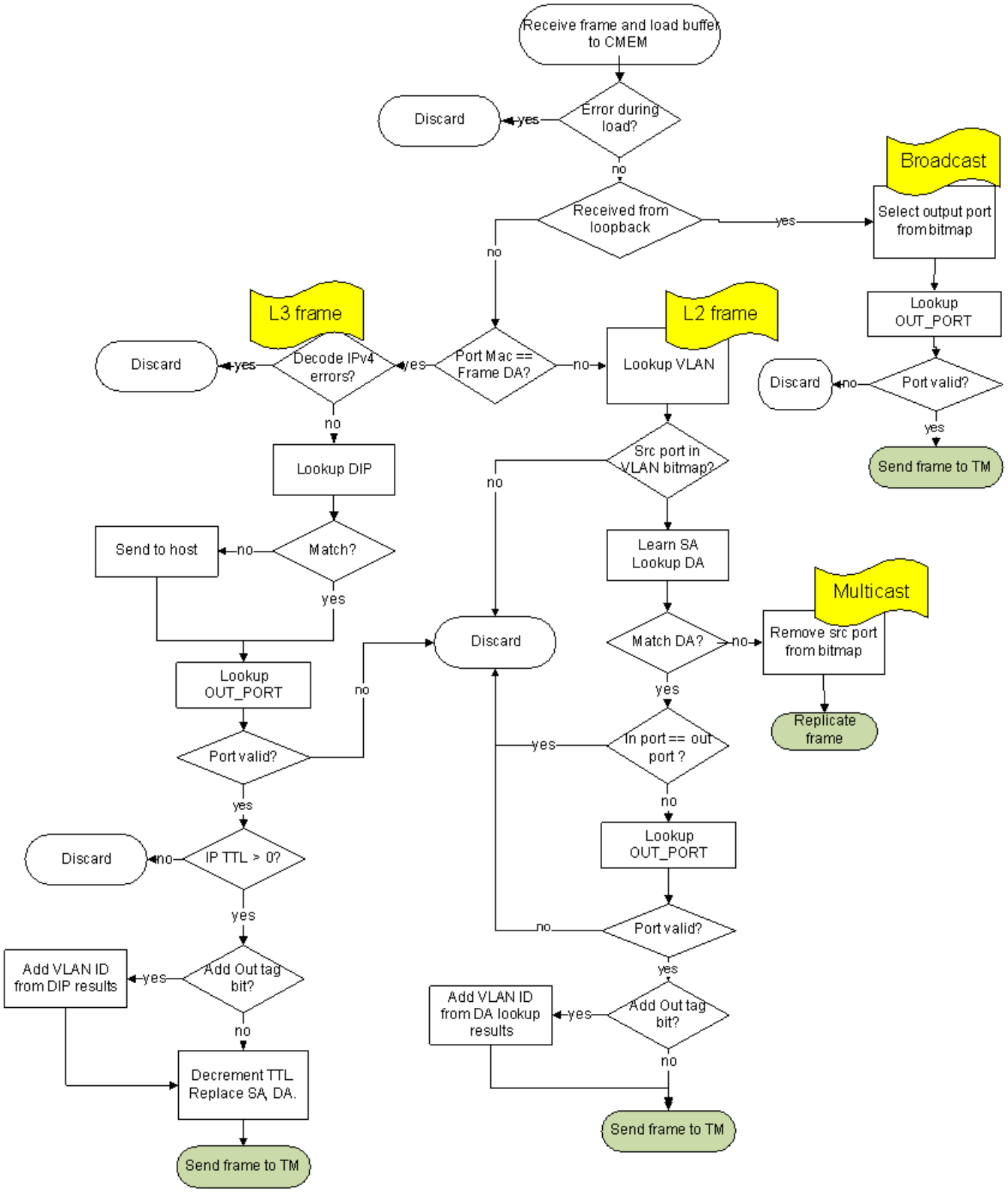
- If source port equals out port from DA result, discard the frame.
- Lookup output port. If port invalid, discard frame.
- Add VLAN tag to frame if needed according to output port lookup.
- Send frame to TM according to QoS algorithm description.

Step f. Else (no match in DA lookup), multicast frame:

- Remove source port from port bitmap received from VLAN lookup.

- Add replication header to frame containing port bitmap, link select for LAG and flow offset for QoS algorithm.
- Replicate frame using TM loopback port.

Figure 4: L23Qos Flow Chart



2.2 Quality of Service Algorithm

In the sample, quality of service (QoS) is provided by the Traffic Manager. To provide the required QoS for a given frame, the application finds the TM queue number that corresponds to the frame. The queue number is found by performing one or two lookups, depending on the frame handling (L2 or L3).

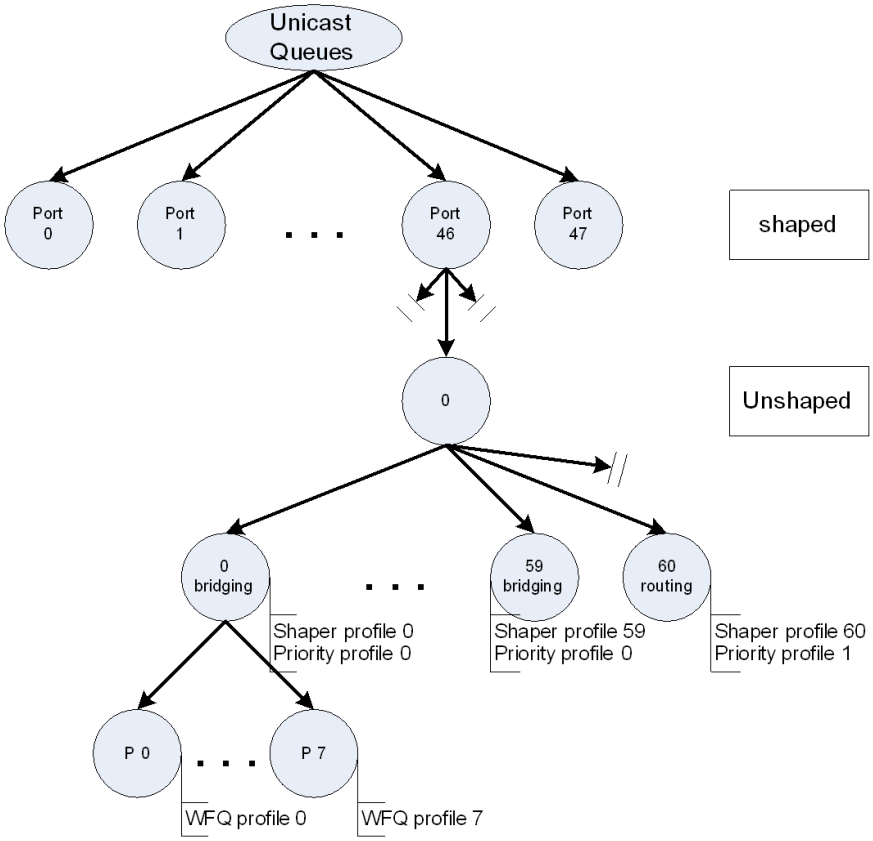
For the L2 switching case, lookup is performed in the "VLAN Structure (table)" and finds the L3 entity corresponding to one of 60 shaping groups of VIDs. Additional lookup in the "OUT-_PORT Structure (table)" gets the base number (L1 entity) corresponding to this port. The TM queue is then calculated by base + offset + index, where index is taken from the frame (VLAN priority).

For the L3 routing case, only the output port parameter is required, so one lookup finds the base, and the TM queue is calculated by base + constant (entity 60) + index, where index is taken from the frame (IP TOS 3 LSB).

2.2.1 Traffic Manager Configuration

All frames pass through the Traffic Managers (either TM0 or TM1).
The general TM topology configuration is shown below:

Figure 5: General TM Topology Configuration



In the TM, there is a 1:1 correlation between L1 and L2. There are 64 L3 entities per L2 subport (only 61 are enabled) and 8 priorities for L4 flows.

The TM has 48 L1 entities (port level) enabled with one subport enabled per port.

2.2.2 Traffic Manager Topology

For L2 switched frames, there is mapping between all VIDs to 60 entities on L3. Each such entity owns a separate shaping profile. In addition, there are 8 priority flows on L4 connected to each L3 entity defining prioritization inside the shaped VLAN.

All L3 Routing frames are mapped to entity #60 on L3 and have 8 priority flows on L4. Each L4 entity is configured with a separate WFQ profile.

Example of WFQ parameters:

WFQ 0:1/1
WFQ 2:16/16
WFQ 3:32/32
WFQ 4:48/48
WFQ 5:64/64
WFQ 6:80/80
WFQ 7:96/96

2.3 Command Line

The application receives a string representing the CTOP processors to run as a command line parameter. The string can be a list of processor IDs separated by commas (e.g. "1,2,3"), an interval of IDs ("0-4"), or a combination thereof ("0,2-4,6,8-10").

The number of processors received represents the number of child processes that will be created. The father application will spawn the needed child processes and will pass each child a unique processor ID from the list. Each child process then binds itself to the proper processor, and obtains its own private resources and mappings. After the work processes creation, the father process waits indefinitely. When the father process receives a shutdown signal, it will kill all the child processes and terminate the application.

If only one processor ID was received, the application will bind itself to that core without creating any child processes. However, if no argument was supplied, the application will run without binding to a specific processor.

3 Supplied Files, Frames and Search Entries

3.1 Input and Configuration Files

The L23Qos application is supplied with two system configurations:

- 96 XFI – 48x 10Gpbs XFI interfaces per side.
- 8 CAUI – 4x 100GE CAUI interfaces per side.

For each system configuration, the config.xml, search_entries.xml and frame files are supplied.

3.1.1 Configuration Files

For each system configuration, only the interfaces used and the frame naming differs.

config.xml

- Interfaces (96XFIs): 96 10Gbps interfaces; 48 XFI ports per side; 12 interfaces per engine.
- Interfaces (8CAUIs): 8 100GE CAUI interfaces; 4 interfaces per side; each interface consume a whole engine.
- PMU: PMU queue 0 in each PMU block is enabled.
- Search: Six search structures are configured (see the "Search Structures" chapter).

search_entries.xml

- This file contains the configuration of the search entries for the six search structures (see the "Frame Formats and Search Entries" section).
- There are 5 test cases and for each test case there is a separate search_entries.xml containing the relevant entries.

3.1.2 Input Frames Supplied

The input frames are provided in *.pcap format, one file per port. These files can be found in the /frames directory. Each test case has its separate folder of PCAP file frames.

3.1.3 Test Cases

There are four test cases.

96 XFI Setup:

- L2 – forwarding frames using L2 switching.
  All incoming traffic from side 0 is forwarded to the same interface on side 1 and vice versa (e.g. from interface 0, side 0 to interface 0, side 1).
  There are 21 different streams per interfaces with a total of 2K streams for the whole setup. Each interface on side 0 receives single tagged traffic where its incoming VLAN range is between 100 – 120.
  Each outgoing frames from side 0 to side 1 is modified and its incoming VLAN is replaced with incoming VLAN_ID+1000.
  From side 1 to side 0 the application receives also single tagged traffic where the VLAN

range is between 1100 – 1120.
Each outgoing frames from side 1 to side 0 is also modified and its incoming VLAN is replaced with incoming VLAN_ID-1000.
All frames on both sides pass through the TM.

- L3 – perform L3 routing operation. Similar to the L2 test case. This test case also has 21 different streams per port. All incoming frames have a DST MAC which is recognized as MY MAC.
  The frame flow is similar to L2. All incoming frames from side 0 are transmitted to the same port on side 1 and vice versa.

All of the streams on both sides pass two steps of modification:

Step 1. Updating the L2 header with MAC addresses.

Step 2. Swapping the VLAN ID of the frame. From side 0 to side 1 – add 1000 to the VLAN ID. from side 1 to side 0 – decrement 1000 from the VLAN ID.

8 CAUI:

- L2 – similar to the L2 test case of 96 XFI setup but with a different setup.
  Each interface has 256 different streams instead of 21.
- L3 – similar to the L3 test case of 96 XFI setup but with a different setup.
  Each interface has 256 different streams instead of 21.

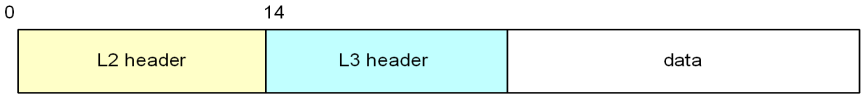
3.2 Frame Formats and Search Entries

This section describes the frame formats to be used with an external traffic generator for the L23Qos sample.

All frames should contain the L2 header.

L3 frames should be IPv4 frames with the following format:

Figure 6: General Header Format



The application processes Ethernet frames.

All ports are configured with following MAC addresses:

Port X: MAC = 00 5A 5A 5A 00 XX

Frames with DA = 00 5A 5A 5A 00 XX that are injected at Port=XX are handled as L3 frames.

Other frames are handled as L2 frames.

This configuration is applied through "IN_PORT Structure (table)" entries.

4 Search Structures

The application uses the search structures defined in the defs.h as below. Structure IDs are defined in an enum as follows:

```
enum search_struct_id
{
                IN_PORT_STR_ID  = 0,           // table
                DA_STR_ID       = 1,           // hash
                DIP_STR_ID      = 2,           // internal tcam
                DIP_INDEX_STR_ID= 3,           // table
                VLAN_STR_ID     = 4,           // table
                OUT_PORT_STR_ID = 5,           // table
};
```

4.1 IN_PORT Structure (table)

This structure determines basic information on the input frame based on the incoming port number.

```
struct in_port_key
{
                uint8_t in_port; // frame's logical source port
};
CASSERT(sizeof(struct in_port_key) == 1);

struct in_port_result
{
                struct {
                uint8_t  control;   // HW response
                uint8_t  is_learn    :1;
                uint8_t  reserved_b7 :7;
                uint16_t default_vlan; // used when C-Vlan is missing from the frame
                union
                {
                   struct ether_addr   my_mac; // port's MAC address
                   struct
                   {
                      uint32_t  my_mac_msb;
                      uint16_t  my_mac_lsb;
                   }__packed;
                };

                uint8_t  reserved_B10_15[6];
                };
};
CASSERT(sizeof(struct in_port_result) == 16);
```

4.2 DA Structure (hash)

This structure maps the MAC destination address to the destination port. It is a dynamic structure that contains both static entries and entries populated by learning on the data path.

```
union da_key
{
   struct
   {
      union
      {
         struct ether_addr  addr;
         uint16_t           addr_word[ETH_ALEN/2];
      };
                    uint16_t vlan_id;
   };
   uint64_t raw_data;
};

CASSERT(sizeof(union da_key) == 8);

union da_result
{
                  struct {
                  uint8_t  control;  // HW response
                  uint8_t  out_port; // logical port to send the frame to
                  uint16_t  out_vlan;
                  uint8_t  reserved_B2_7[4];
                  };
};
CASSERT(sizeof(union da_result) == 8);
```

4.3 DIP Structure

This internal TCAM structure is used in the L3 routing path to resolve the output port, VLAN tag, and destination MAC address per destination IP.

```
struct dip_itcam_key
{
 unsigned rsv1 :32;
 unsigned rsv2 :16;
 uint32_t dest_ip ; // destination IP from the frame
}__packed;

CASSERT(sizeof(struct dip_itcam_key) == 10);

struct dip_itcam_result
{
```

```
 struct ezdp_lookup_int_tcam_result result;
};
```

4.4 DIP_INDEX Structure (table)

This structure is a helper structure for "DIP Structure", since it consists of only 4 bytes of result. The result received from the DIP Structure is used as a key here.

```
struct dip_index_key
{
                uint16_t index; // as found in DIP result
};
CASSERT(sizeof(struct dip_index_key) == 2);

struct dip_index_result
{
                uint8_t  control;  // HW response
                uint8_t  out_port; // logical port to send the frame to
                uint16_t vlan_tag; // to be used in case adding VLAN is needed
                uint8_t  reserved_B4_5[2];
                union
                {
                   struct ether_addr  da_addr; // replace current DA with this
                   uint16_t           addr_word[ETH_ALEN/2];
                };
                uint8_t  reserved_B12_15[4];
};

CASSERT(sizeof(struct dip_index_result) == 16);
```

4.5 VLAN Structure (table)

This structure is used to retrieve which ports participate in a VLAN group.
In case of broadcast, TM flow ID and WRED profile will be taken from this lookup result.

```
struct vlan_key
{
                uint16_t  vlan_id; // taken from the frame or default if none exists
};
CASSERT(sizeof(struct vlan_key) == 2);

struct port_bitmap
{
                uint32_t side0_0;  // ports belonging to the VLAN (first 32 ports of side 0)
                uint32_t side0_32; // ports belonging to the VLAN (ports 32-63 of side 0)
                uint32_t side1_0;  // ports belonging to the VLAN (first 32 ports of side 1)
                uint32_t side1_32; // ports belonging to the VLAN (ports 32-63 of side 1)
};
```

```
struct vlan_lookup_result
{
                uint8_t  control;        // HW response
                uint8_t  reserved_B1;
                uint16_t tm_index;       // part of Qos algorithm (added to FlowID)
                struct port_bitmapbitmap; // contain all ports in a VLAN
                union{
                uint32_t        flow_id_word;
                struct{
                                unsigned  reserved_12b:12;
                                unsigned  flow_id:(EZDP_JOB_TX_INFO_FLOW_ID_SIZE +
                                             EZDP_JOB_TX_INFO_SIDE_SIZE);
                };
                };
                ezframe_wred_profile_t  wred_profile;
                uint32_t  reserved_4B;
};
CASSERT(sizeof(struct vlan_lookup_result) == 32)
```

4.6 OUT_PORT Structure (table)

This structure is used to map the destination logical port ID with the various fields required for sending to this port, such as its TM flow ID and WRED profile.

```
struct out_port_key
{
                uint8_t  out_port; // logical out port
};
CASSERT(sizeof(struct out_port_key) == 1);

union out_port_control
{
                uint32_t raw_data;
                struct
                {
      unsigned  /*reserved*/      : EZDP_LOOKUP_PARITY_BITS_SIZE;
      unsigned  /*reserved*/      : EZDP_LOOKUP_RESERVED_BITS_SIZE;
      unsigned  memory_error      : 1; // might be turned 'on' by HW
      unsigned  out_port_invalid  : 1; // if '1' - discard the frame
      unsigned  add_out_tag       : 1; // if '1' - add C-Vlan
      unsigned  /*reserved*/      : 1;
                };
};

struct out_port_result
{
                /*byte 0-3*/
```

```
                union out_port_controlcontrol_data;
                /*byte 4 -7 */
                union{
                uint32_t        flow_id_word;
                struct
{
                                unsigned  reserved_12b:12;
                                unsigned  flow_id:(EZDP_JOB_TX_INFO_FLOW_ID_SIZE +
                                  EZDP_JOB_TX_INFO_SIDE_SIZE);
                };
                };
                /*byte 8 -11 */
                ezframe_wred_profile_twred_profile;
                /*byte 12 -15 */
                unsigned                                :32;
};

CASSERT(sizeof(struct out_port_result) == 16);
```

5 Running the L23Qos Application

5.1 Compiling the Application

This section provides detailed information on how to compile the L23QoS Application.

5.1.1 Compiling with Makefile

Step 1. Go to the directory /samples/l23qos/l23qos_dp

Step 2. Build the l23qos_dp application using the command:

```
EZDK_BASE={base EZdk path} EZDK_PLATFORM=linux_arc make
```

5.1.2 Launch from Linux Shell on a Real Chip

Step 1. Open a shell on your Linux system.

Step 2. Use one of the below scripts to run an L23QoS sample test case on your box:

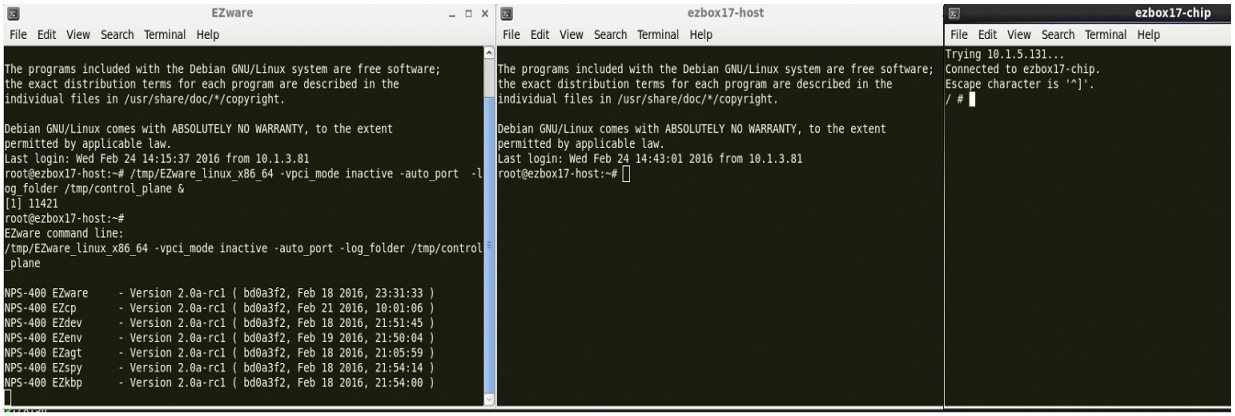
- Runs L2 test case on the 96XFI port configuration.
  ./samples/l23qos/l23qos_target/launches/open_npu/run_l23qos_target_rc_96xfi_l2.sh
- Runs L3 test case on the 96XFI port configuration.
  ./samples/l23qos/l23qos_target/launches/open_npu/run_l23qos_target_rc_96xfi_l3.sh
- Runs L2 test case on the 8CAUI port configuration.
  ./samples/l23qos/l23qos_target/launches/open_npu/run_l23qos_target_rc_8caui_l2.sh
- Runs L3 test case on the 8CAUI port configuration.
  ./samples/l23qos/l23qos_target/launches/open_npu/run_l23qos_target_rc_8caui_l3.sh

Step 3. Use the following command line in order to run a test case:

```
./samples/l23qos/l23qos_target/launches/open_npu/run_l23qos_target_rc_96xfi_l2.sh
.  [base EZdk path] [sample_dp][host_ip][chip ip]
```

Step 4. The script will start logging to screen the action it performs and open three new terminal windows:

- EZware - running the driver - do not close this window until finish testing.
- HOST - a terminal with ssh connection to the host.
- CHIP - a terminal with telnet connection to the chip.



Step 5. In the CHIP terminal, the sample bin file is located under the /tmp folder. The sample is not automatically run and the user needs to manually run it, e.g. /tmp/dp_app.