# Intel® IXP400 Software: VLAN and QoS Application Version 1.0

## Programmer's Guide

**September 2004**

Document Number: 301925-001

# Contents

# Figures



# Tables

# Revision History

| Date | Revision | Description |
|---|---|---|
| September 2004 | 001 | Initial release. |

**This page intentionally left blank.**

# 1.0 Introduction

## 1.1 Scope and Purpose

The purpose of this document is to provide high-level technical design information for the Intel® IXP400 Software VLAN and QoS Application. Based on Intel® IXP400 Software v1.4, this example code is provided to implement IEEE 802.1Q VLAN (Virtual Local Area Networks), IEEE 802.1p User Priority to Traffic Class (TC) mappings, and Ingress Quality of Services (QoS) functionality for IPv4 traffic using the IXP400 software.

This document covers the high-level functionality of the various modules, and describes their behavioral links. For a more complete understanding, you should review the API reference information provided in Section 6.0, "API Reference" on page 35, and review the VLAN and QoS Application v1.0 user interface (as described in the Release Notes) and source code.

It is assumed that you are familiar with IEEE 802.1D Ethernet bridging and IEEE 802.1Q/p VLAN and Priority functionality.

## 1.2 Acronyms

| | |
|---|---|
| FIFO | First In, First Out |
| ID | Identification |
| IEEE | Institute of Electrical and Electronics Engineers |
| IO | Input / Output |
| IOCTL | I/O Control |
| LAN | Local Area Network |
| MAC | Media Access Controller |
| NPE | Network Processing Engine |
| OS | Operating System |
| TC | Traffic Class |
| QoS | Quality of Service |
| VLAN | Virtual LAN |

## 1.3 Related Documents

Additional Intel documents listed below are available from your field representative or from the following Web site:

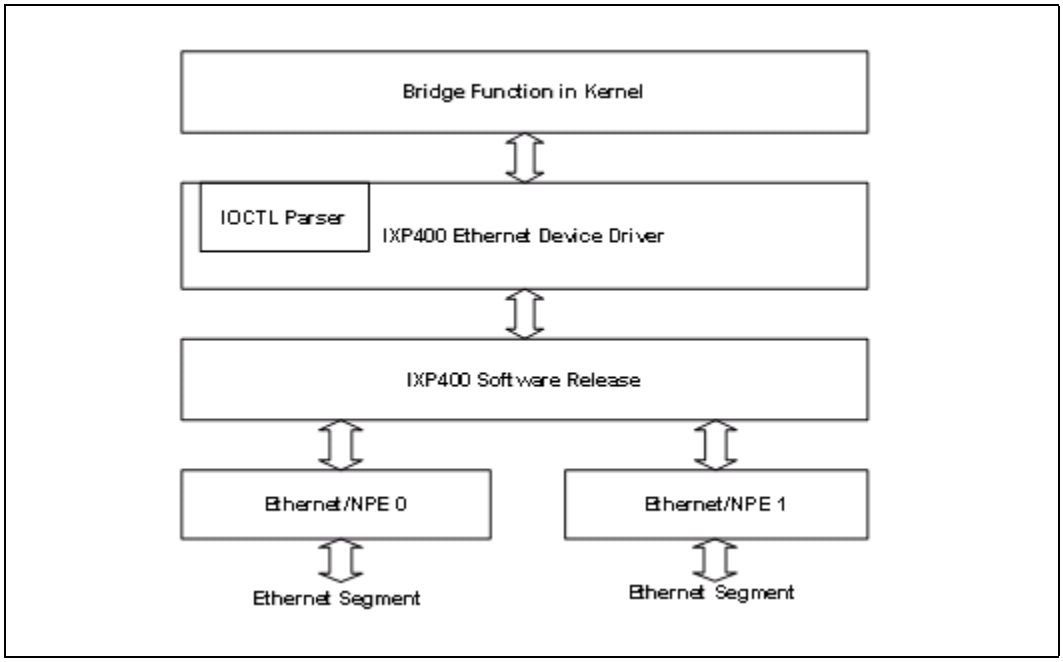
http://www.intel.com/design/network/products/npfamily/docs/ixp4xx.htm

| Document Title | Document # |
|---|---|
| *Intel® IXP400 Software: VLAN and QoS Application Version 1.0 Programmer's Guide* | 301925 |
| *Intel® IXP400 Software: VLAN and QoS Application Version 1.0 Release Notes* | N/A |
| *Intel® IXP400 Software Release 1.4 Software Release Notes* | N/A |
| *Intel® IXP400 Software Programmer's Guide* (for Release v1.4) | 252539-005 |
| *Intel® IXP400 Software Specification Update* | 273795 |
| IEEE Standards (IEEE Std 802.1D-1998) for Local Area and Metropolitan Networks, Media Access Control (MAC) Bridge | N/A |
| IEEE Standards (IEEE Std 802.1Q-1998) for Local Area and Metropolitan Networks, Virtual Bridged Local Area Networks | N/A |
| IEEE Standards (IEEE Std 802.1p-1998) for Traffic class expediting and dynamic multicast filtering | N/A |

# 2.0 Software Architecture and High-Level Design

As depicted in Figure 1, the software architecture of Intel® IXP400 Software VLAN and QoS Application is designed to integrate with the IXP400 software.

**Figure 1. IXP400 Software and Ethernet Device Driver Overview**



The Intel® IXP4XX Product Line of Network Processors and IXC1100 Control Plane Processor contain Network Processing Engines (NPEs), which provide physical connectivity and processing of data to various interfaces. One function of the IXP400 software is to provide OS and upper-level applications access to these interfaces via a set of APIs. In the case of the VLAN and QoS Application v1.0, the two Ethernet NPE ports are the two physical links of an Ethernet bridge. The Ethernet device driver is the OS-specific code that provides access to these NPEs via the services of the IXP400 software.

The IEEE 802.1Q/p and QoS functionality is provided by a set of software modules for 802.1Q VLAN and QoS (including 802.1p Priority Mapping and Ingress QoS). These modules interface with the IXP400 software, the OS-specific device driver for the NPE ports, and the OS-specific bridging software.

The modules do contain a minimal amount of OS-dependent code. When OS-specific code is used, it is enclosed by a compiler definition typically passed through to the makefiles from the IXP400 software build system.

**Figure 2. Software Architecture with the VLAN and QoS Application v1.0**

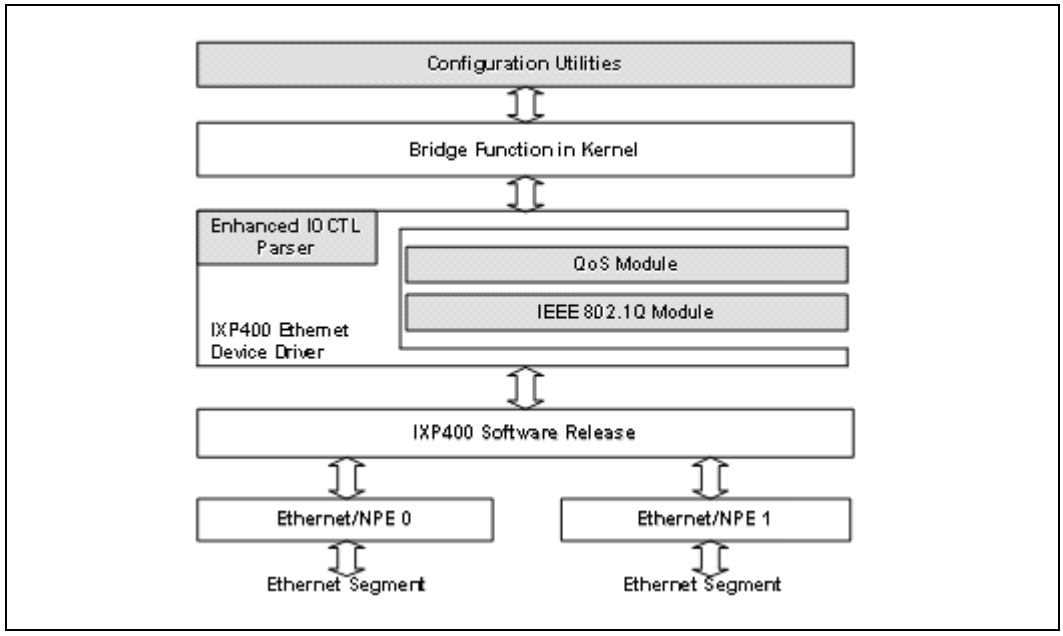


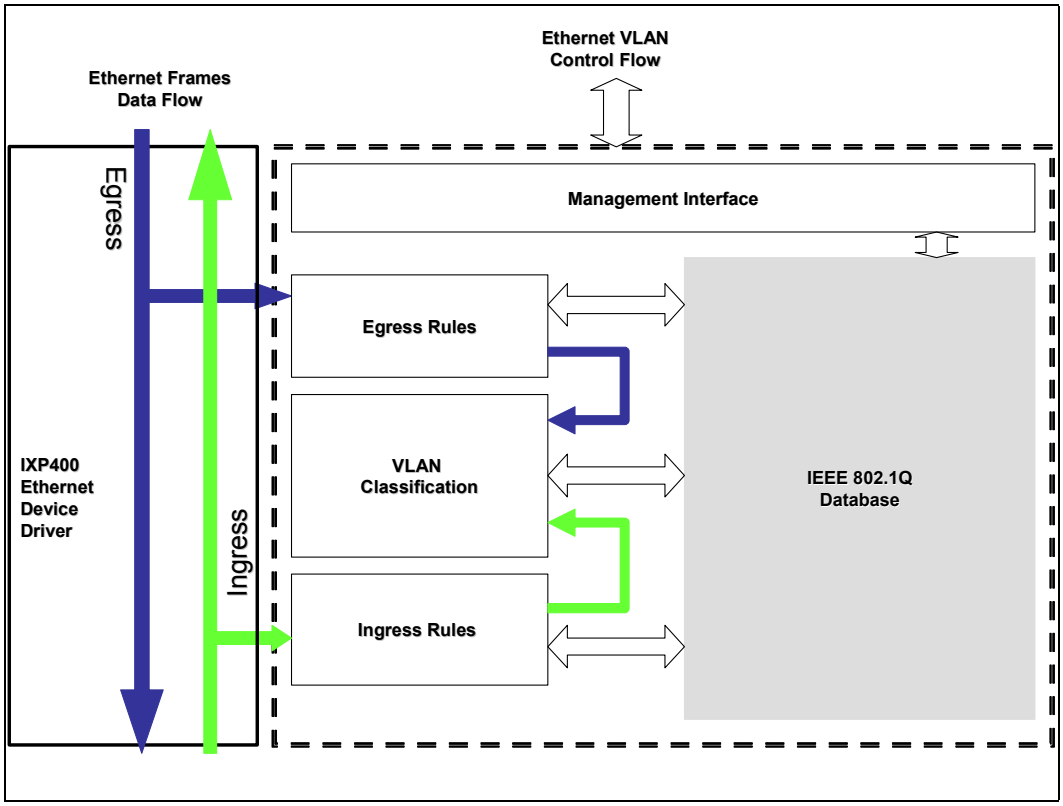

Figure 2 closely resembles Figure 1, but includes the VLAN and QoS Application v1.0. The two modules, one for 802.1Q VLAN and another for QoS processing, are inserted into the data path of the system. The Ethernet device driver uses these modules when VLAN-capable Ethernet frames are received or need to be sent.

The VLAN and QoS Application v1.0 also provides control path capabilities. The Ethernet device driver's IOCTL parser is enhanced to recognize and execute the additional VLAN and QoS functionality.

# 3.0 802.1Q VLAN Module

This module implements the IEEE 802.1Q VLAN functionality. The module includes five software sub-components, which are briefly described below; later sections provide more sub-component detail. Ingress Rules, Egress Rules, and VLAN Classification deal with frame processing, the Database records all information supported by 802.1Q VLAN module, while the Management Interface deals with configuration for each component and provides public APIs for external modules. The general flow, shown in Figure 3, is described below.

**Figure 3. 802.1Q VLAN Module – Component View**




**LAN Module Sub-Components**

- Ingress Rules

  Two Ingress Rule filterings are supported: Acceptable Frame Types and Ingress VLAN Membership filterings. For Acceptable Frame Types filtering, it determines if received frames are “VLAN-tagged”, “priority-tagged” or “non VLAN tagged”. Frames are discarded if the reception port is not allowed to receive these types of frames. For Ingress VLAN Membership filtering, the frames are discarded if the port of the received frames is not in the member set of VLAN group.

- VLAN Classification

  Determines VLAN Identification (VID) and User Priority of received frames (in ingress path) and transmission frames (in egress path).

- Egress Rules

  Two major features are supported: Egress VLAN Membership filtering and Rebuild Packet Header. The function of the Egress VLAN Membership filtering is the same as Ingress VLAN Membership filtering, except that it executes at the egress port. Rebuild Packet Header supports the ability to determine if transmission frames should be tagged or untagged, and then adds/removes/modifies the VLAN-tag header for the outbound frames.

- Management Interface

  Interface for maintaining database and public APIs.

- Databases

  Records all information and rules for the 802.1Q VLAN module. Three sub-databases are supported: Port database, VLAN database and Classification Rules database. The port database is for port-related information such as PVID (Port VLAN Identification) and Acceptable Frame Types parameter. VLAN database is for VLAN information such as VLAN membership, and Egress port attributes (tagged or untagged). Classification Rules database is for classification rules such as MAC-based classification rules and Protocol (Layer 3/4)-based classification rules.

**Private Frame Buffer Memory**

16 or 32 bytes of extra memory is reserved for each frame buffer that is used by the VLAN module for storing per-frame VID and user priority information. Additionally, this private memory can be used to append the VLAN header information on to a transmission frame. This private memory is transparent to the IXP400 software.

**Receive Path**

For inbound Ethernet Frames, the device driver calls the API of the Ingress Rules component for VLAN ingress processing. The Ingress Rules component analyzes the frame type (VLAN-tagged, priority-tagged or non VLAN-tagged, see Figure 4) of received frames and commences Acceptance Frame Type Check filtering. Frames are discarded if their frame types are not allowed on the reception port. If frames pass the Acceptance Frame Type Check filtering, the Ingress Rules component then calls the services of the VLAN Classification component to determine VLAN Identification (VID) and user priority of received frames. Next, the Database component is queried to get the (port) member set of the detected VLAN group and decide if the reception port is in the member set of that VLAN group or not. Frames are discarded if the reception port is not in the member set of the detected VLAN group. For frames that pass the VLAN Ingress Rules (Acceptable Frame Type Check and VLAN Membership filtering), the VID and user priority are saved into the private area and frames are relayed to the kernel for the bridging process. In addition to the VID and User Priority data, the module also calculates a signature and checksum and stores this information in the private area to help ensure data intregrity.

**Transmit Path**

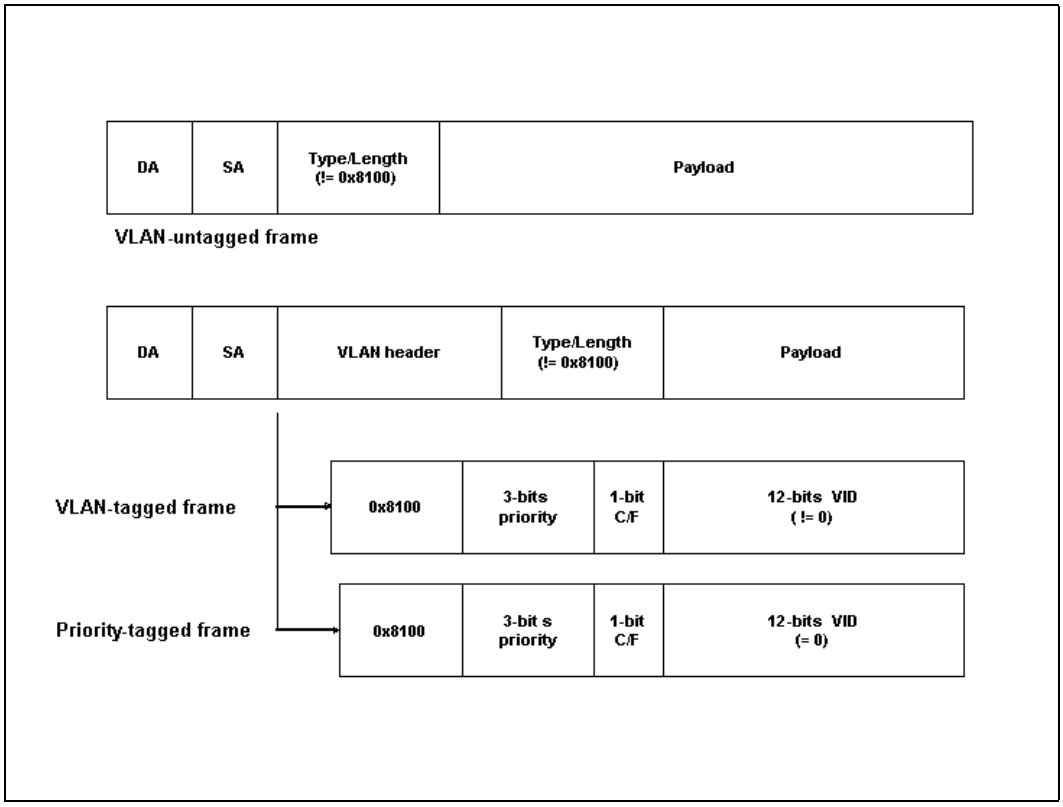
For outbound Ethernet frames, the device driver calls the API of the Egress Rules component for VLAN Egress processing if the Egress Rules component determines frames are bridged from the other NPE/Ethernet port or from an upper-layer application. If frames come from the bridge, ingress-determined VLAN Identification (VID) and user priority (both saved in the private area) are retrieved. Otherwise, the Egress Rules component calls the services of VLAN classification component to determine the VID and user priority of transmission frames. When VID and user priority of transmission frames are determined, the Database component provides the (port) member set of the VLAN group and decides whether or not the transmission port is in the member set of VLAN group. Frames are discarded if the transmission port is not in the member set of the VLAN group. If the transmission port is in the member set, egress attributes (VLAN-tagged or

VLAN-untagged) of the transmission port in the VLAN group and the frame type of the outbound frames are used to determine whether or not the frame header of transmission frames should be rebuilt (insert or remove VLAN header). After completing all egress processes, the device driver calls the IXP400 software APIs required to transmit the frames.

## 3.1 Ingress Rules Component (ixVlanIngress)

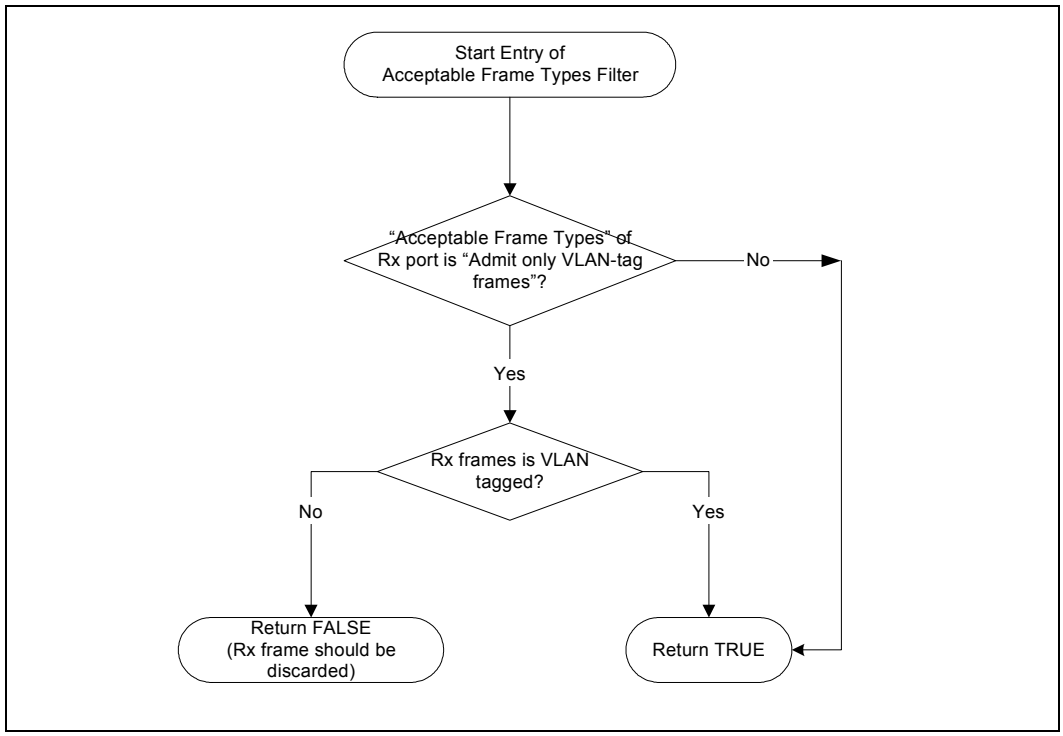
This component supports the functionality of IEEE 802.1Q Acceptable Frame Types and Ingress VLAN Membership filtering. In the IEEE 802.1Q standard, three frame types are defined — VLAN tagged, priority-tagged, and non VLAN-tagged.

**Figure 4. 802.1Q Frame Types**



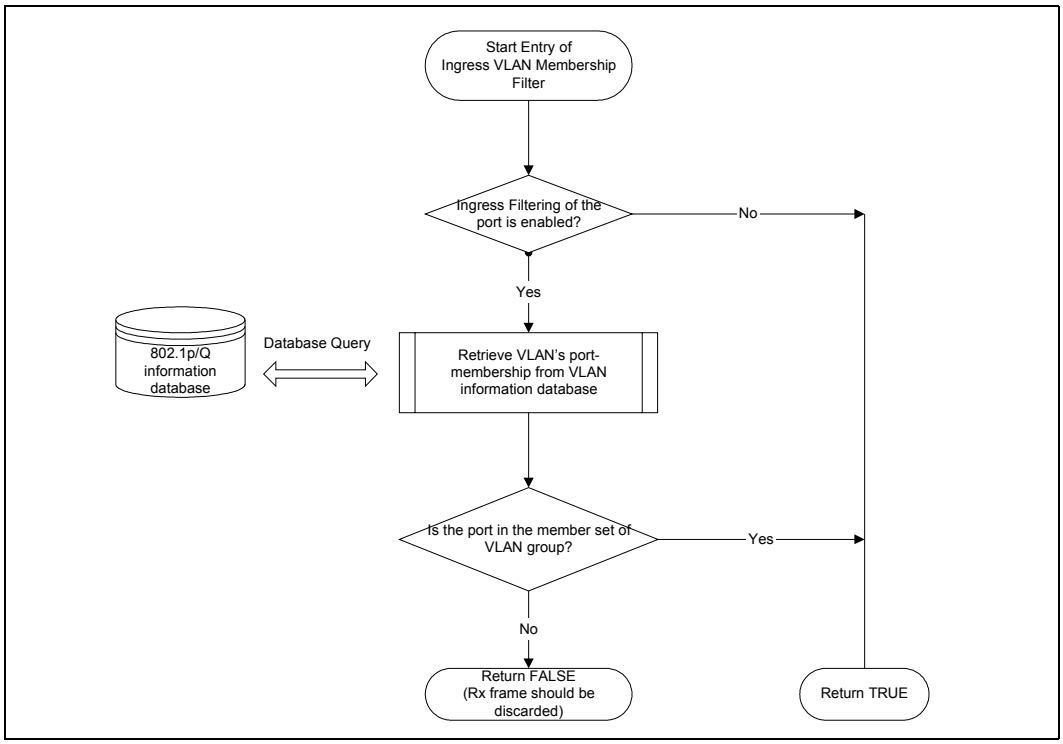

The Acceptable Frame Types parameter associated with each port controls the reception of the types of frames on that port. Valid values for this parameter are: “Admit Only VLAN-tag Frames” and “Admit All Frames”. If it is set to “Admit Only VLAN-tag Frames”, any frames received on that port which do not contain VID tagging information (i.e., untagged frames and priority-tagged frames) are discarded.

**Figure 5. Flow Diagram for Acceptable Frame Type Filtering**




Ingress VLAN Membership filtering discards any frames received on that port whose VLAN group does not include that port in its member set.

**Figure 6. Flow Diagram for Ingress VLAN Membership Filtering**




### 3.1.1 External Interactions and Dependencies

The Ethernet device driver utilizes this component to perform IEEE 802.1Q Ingress Rules (Acceptable Frame Types and VLAN Membership) filtering. It is heavily dependent on VLAN information in the Database component. Depending on whether or not Ingress Filtering is enabled, information about Acceptable Frame Types parameters (and the member set of the VLAN group) is queried from the Database component.

### 3.1.2 Key Assumptions

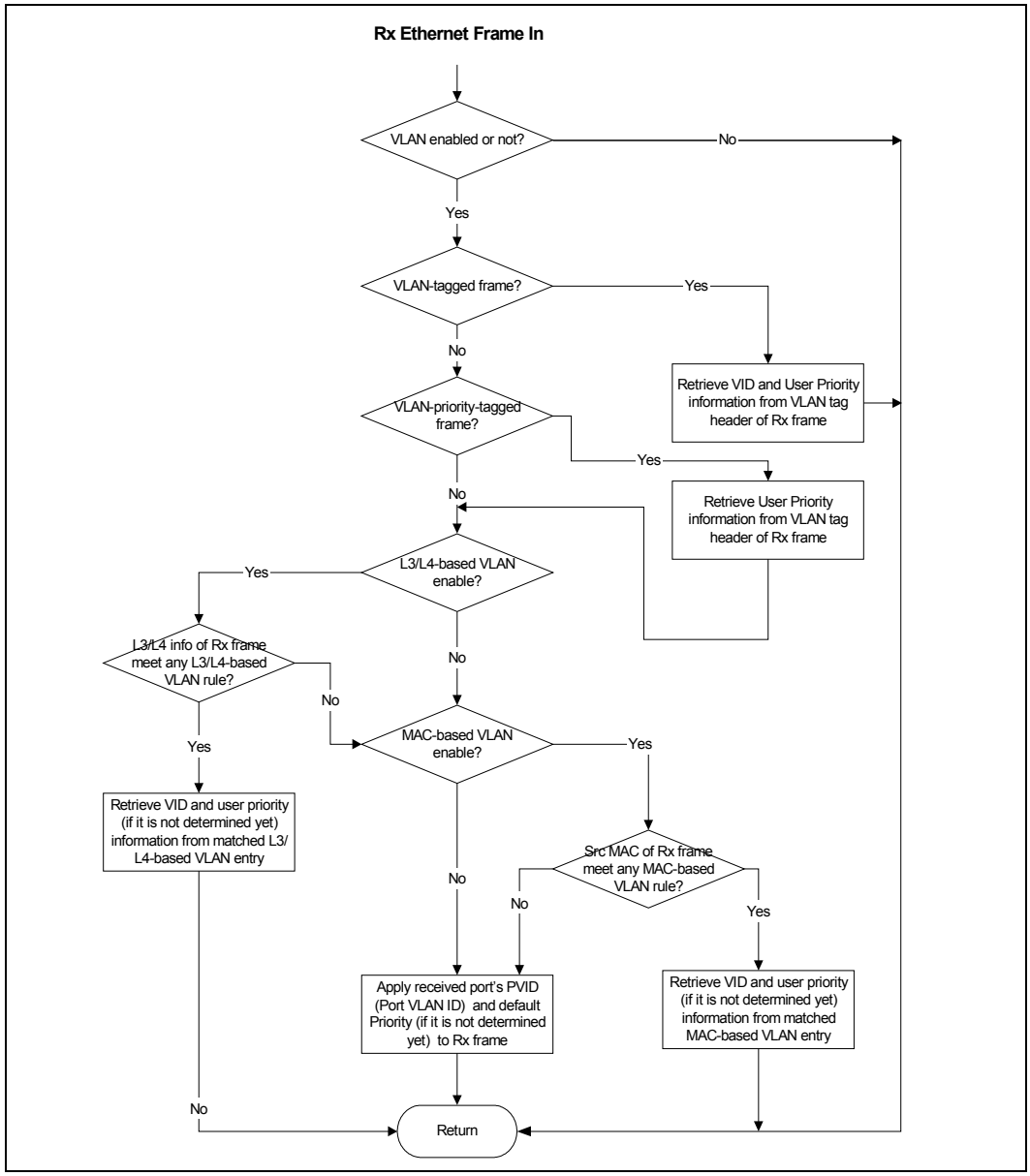
- The Database component in VLAN module is initialized and available for query.
- Default value of Acceptable Frame Types parameter for all ports is “Admit All Frames”.
- “VLAN_DEBUG” definition is used to determine whether or not the code is operating under a debugging environment. If “VLAN_DEBUG” is not defined, all input parameters to this component should be valid. For performance reasons, under normal operation no routines in this component perform any input parameter checking.

## 3.2 VLAN Classification Component (ixVlanClassification)

VLAN Classification component is used to determine VLAN Identification (VID) and User Priority of reception/transmission frames in accordance with established classification rules. Four kinds of classification rules are supported: 802.1Q tag-based, port-based, MAC-based and Protocol (Layer 3/4) -based classifications. 802.1Q tag-based classification determines VID and priority

from the VLAN-tag header of received frames. Port-based classification uses the reception port of frames to decide VID and priority. MAC-based classification uses source MAC address, and Protocol (Layer 3/4) -based classification uses information in IP/UDP/TCP headers to determine a frames VID and priority.

**Figure 7. Flow Diagram for VLAN Classification**




## 3.2.1 External Interactions and Dependencies

The Ethernet device driver utilizes the VLAN Classification component to perform Ingress and Egress VLAN Classification. The driver also utilizes services in the Database component to determine if any classification rules should be applied to frames.

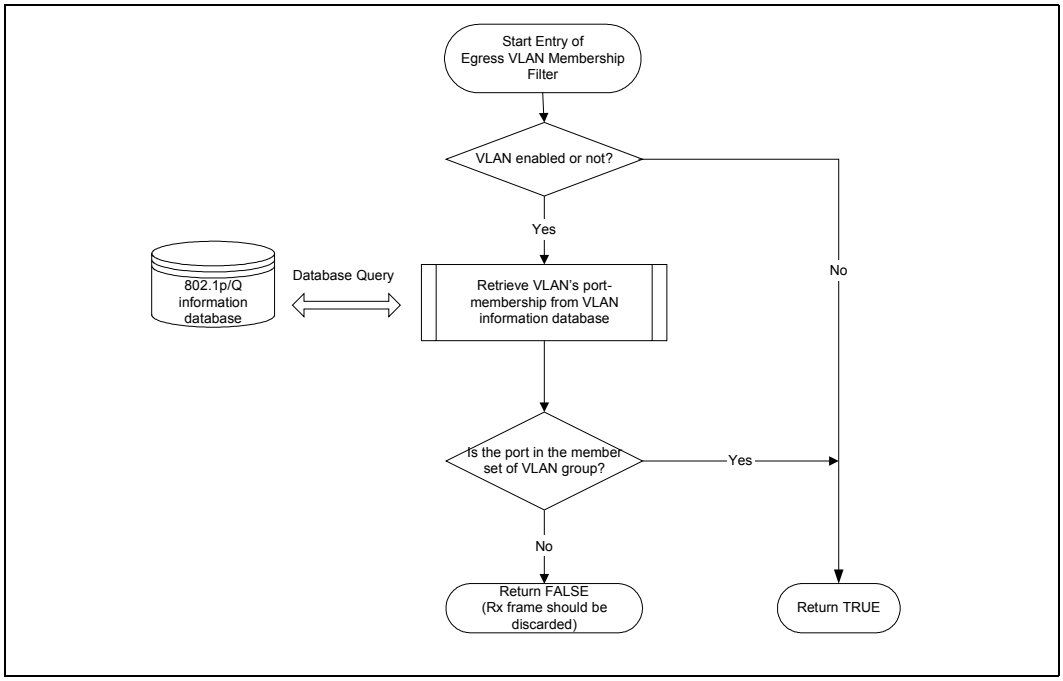
### 3.2.2 Key Assumptions

- "VLAN_DEBUG" definition is used to determine whether or not the code is operating under a debugging environment. If "VLAN_DEBUG" is not defined, all input parameters to this component should be valid. For performance considerations, under normal operation no routines in this component perform any input parameter checking.

## 3.3 Egress Rules Component (ixVlanEgress)

This component provides functionality for IEEE 802.1Q VLAN Egress Rules. Two features are supported: Egress VLAN Membership filtering and Rebuild the Frame Header. Egress VLAN Membership filtering discards frames whose transmission ports are not present in a frame's VID member set.

**Figure 8. Flow Diagram for Egress VLAN Membership Filtering**




In the IEEE 802.1Q Standard, on a given link a VLAN-aware bridge can transmit untagged frames for some VLANs and VLAN-tagged frames for others, but cannot transmit both formats for the same VLAN. A feature is provided for adding, modifying, or removing VLAN tag headers from transmission frames in accordance with tagging requirements on egress for each port. This behavior is described in Figure 9 and Table 1.

**Figure 9. Flow Diagram for Rebuilding the Frame Header**

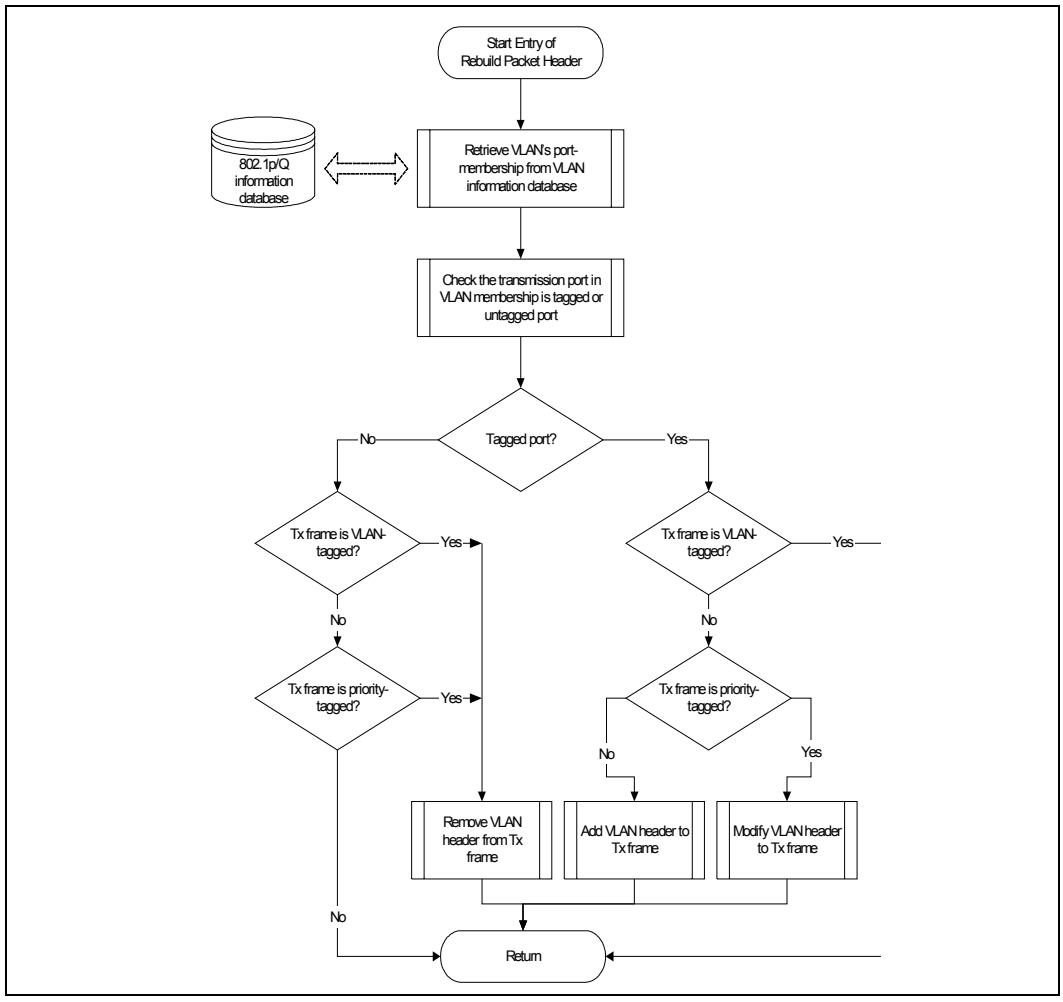



**Table 1. Rules for Rebuilding Frame Headers**

| Transmit Port Transmits Frame as: | Receive Port Receives frame as: | | |
|---|---|---|---|
| | VLAN-Tagged | Priority-Tagged | Untagged |
| untagged | Remove tag header | Remove tag header | N/A |
| VLAN tagged | N/A | Modify VLAN header | Add VLAN tag header |

## 3.3.1 External Interactions and Dependencies

The NPE Ethernet device driver utilizes this component to perform 802.1Q Egress Rules functions. It is heavily dependent on VLAN information in the Database component. Information about whether Egress Filtering is enabled or not and the member set of VLAN group are queried from the Database component.

### 3.3.2 Key Assumptions

- Database component in VLAN module is initialized and available for query.
- Frames that may require the insertion of VLAN-tag headers should reserve enough memory space for VLAN-tag header. If, in the original transmission buffer, not enough memory is available, the driver will allocate memory as needed.
- "VLAN_DEBUG" definition is used to determine whether or not the code is operating under a debugging environment. If "VLAN_DEBUG" is not defined, all input parameters to this component should be valid. For performance considerations, under normal operation no routines in this component perform any input parameter checking.

## 3.4 Database Component (ixVlanDb)

This component contains all VLAN information for 802.1Q VLAN operations. There are three categories (sub-database) of information in this database: Port Database, VLAN Database, and Classification Database.

**Port Database**

This database contains information about port configurations as follows:

- PVID (Port VLAN Identification) and Default User Priority
- Status of Ingress Filtering (enable or disable)
- Acceptable frame types (AdmitAllFrames or AdmitOnlyVlanTaggedFrames)

**VLAN Database**

This database contains the following information about port configurations:

- VLAN function is enabled or not.
- Existing VLAN groups.
- VLAN (port) membership
- Egress attributes (VLAN-tagged or VLAN-untagged) of ports in VLAN group.

  ***Note:*** VLAN membership includes the ports that belong to a VLAN group, egress attributes (VLAN-tagged or VLAN-untagged) of a port in the VLAN group, etc. Please reference the IEEE 802.1Q standard for more information.

**Classification Rules Database**

This database contains the following classification rules:

- Rules for MAC-based classification
- Rules for Protocol (Layer 3/4) -based classification.

There are several general characteristics and functions provided by the Database component:

- Services to configure and query information of Port Database, VLAN Database, and Classification Rules Database.
- Up to 32 VLAN groups are supported simultaneously.

- Up to 16 MAC-based classification rules and up to 16 Protocol (Layer 3/4) -based classification rules are supported, for up to 32 simultaneous rules.

## 3.4.1 External Interactions and Dependencies

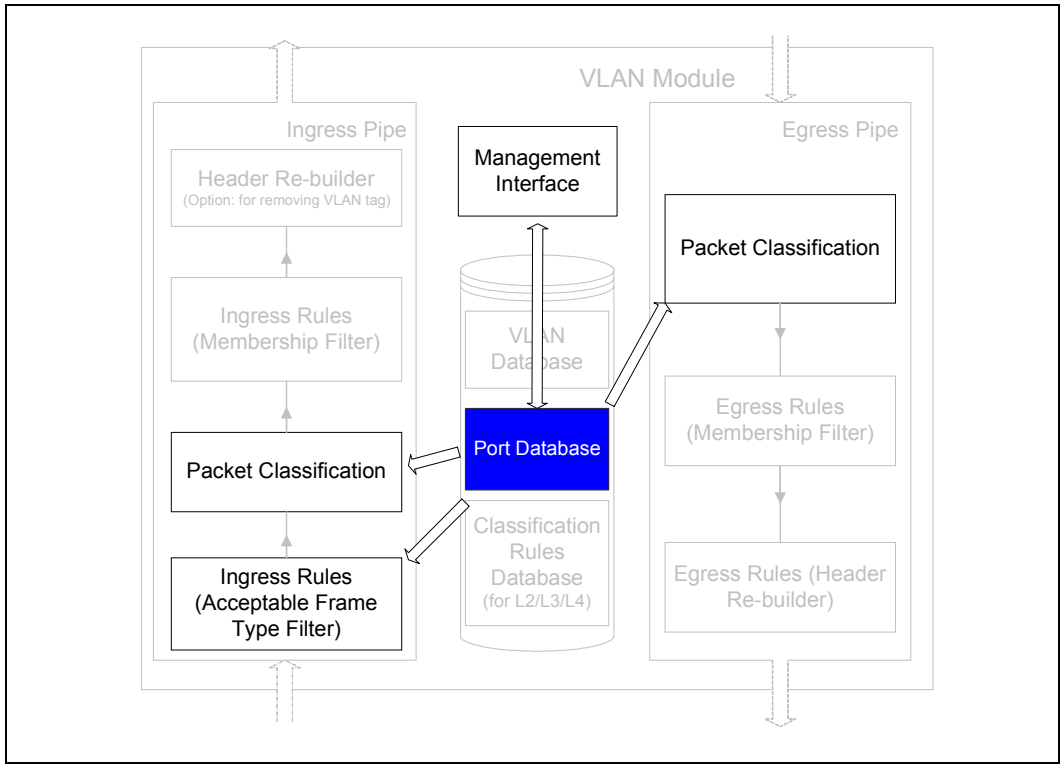
The Database component houses critical information attributes used by other sub-components of the VLAN module. Those interactions (detailed in this section) depend on which sub-database is used.

### 3.4.1.1 Port Database

Depending on the desired service, the Port database is used by the following sub-components:

- Ingress Rules: Ingress Filtering and Acceptable Frame Types attributes for each port.
- Packet Classifier: The PVID and Default User Priority for each port.
- Management Interface: Supports interactions from the component APIs.

**Figure 10. Port Database Dependencies**



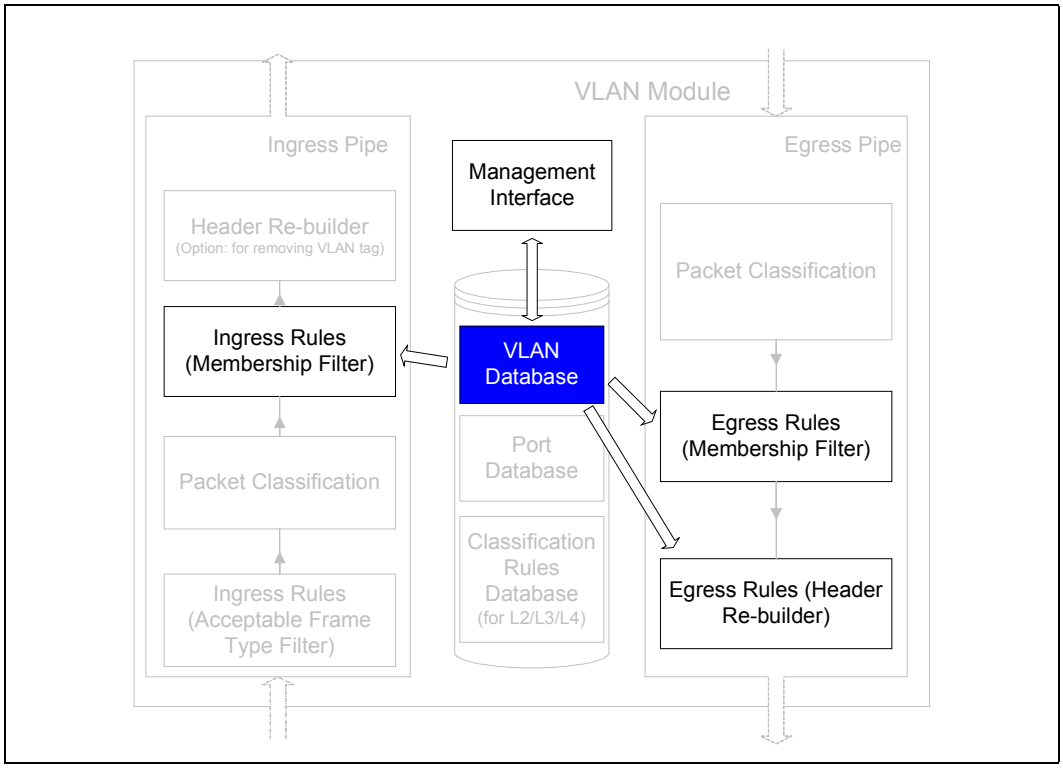

### 3.4.1.2 VLAN Database

For the services described, the VLAN database is used by the following sub-components:

- Ingress Rules: VLAN port membership
- Egress Rules: VLAN port membership. Port attributes (tagged or untagged) for egress ports in a VLAN group.

- Management Interface: Supports interactions from the component APIs

**Figure 11. VLAN Database Dependencies**

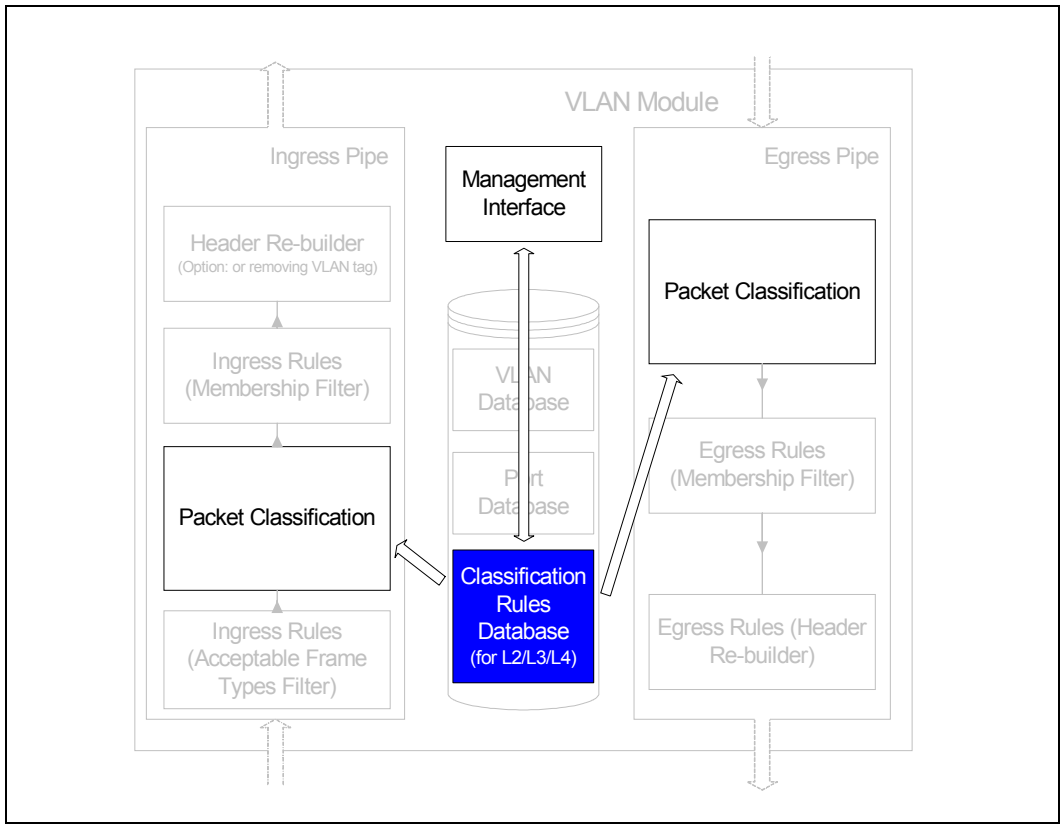



## 3.4.2 Classification Rules Database

Depending on the desired service, the Classification Rules database is used by the following sub-components:

- VLAN Classification: VID and User Priority of frames.
- Management Interface: Supports interactions from the component APIs.

**Figure 12. Classification Rules Database**




### 3.4.3 Key Assumptions

- "VLAN_DEBUG" definition is used to determine whether or not the code is operating under a debugging environment. If "VLAN_DEBUG" is not defined, all input parameters to this component should be valid. For performance considerations, under normal operation no routines in this component perform any input parameter checking.

## 3.5 Management Interface Component (ixVlanMgmt)

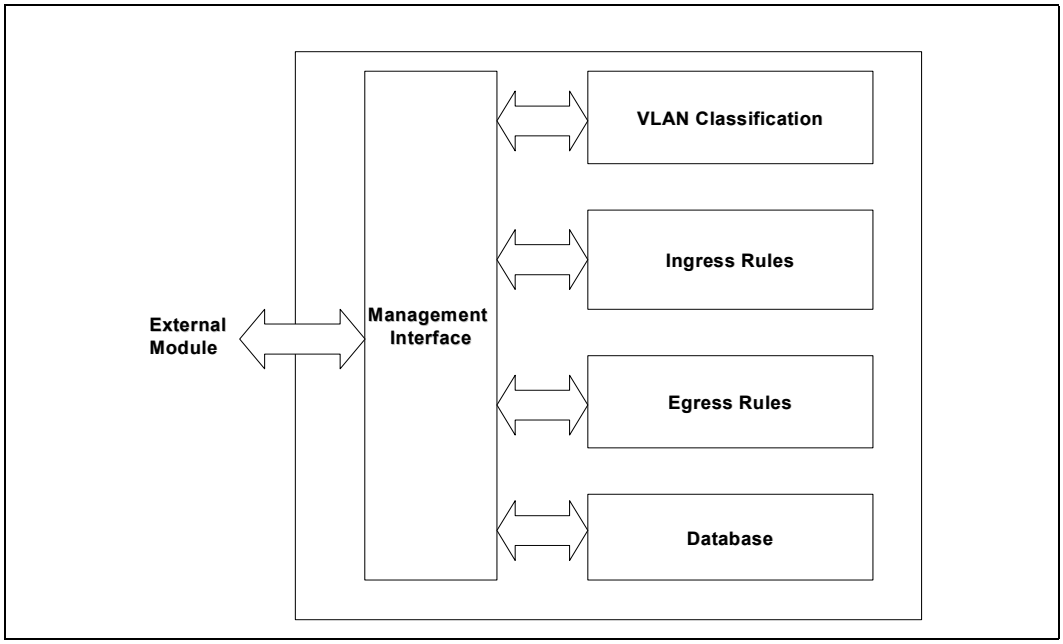
This component provides a unique interface (i.e., control path) for external modules to configure the behavior of the 802.1Q VLAN module. For example, the IOCTL parser in the Ethernet device driver should utilize this interface to access services in 802.1Q VLAN module. Direct accesses to services (or APIs) in other components in this module are not supported.

The features provided by the Management Interface component are:

- Enable & Disable 802.1Q VLAN function.
- Add & Delete VLAN groups.
- Assignment VLAN membership and associated (tagged/untagged) attributes of egress ports.
- Set port's PVID and Default User Priority.

- Configure Acceptable Frame Types filtering of reception port.
- Enable & Disable Ingress (VLAN) Membership filtering.
- Enable & Disable MAC-based VLAN Classification.
- Enable & Disable Protocol (Layer 3/4) -based VLAN Classification.
- Configure MAC-based VLAN Classification Rules.
- Configure Protocol (Layer 3/4) -based VLAN Classification Rules.

**Figure 13. Management Interface Interactions**




### 3.5.1 Key Dependencies

Regardless of whether or not “VLAN_DEBUG” is defined, all routines in this component perform necessary input parameter checking.

# 4.0 802.1p User Priority and QoS Module

This purpose of this software module is to implement IEEE 802.1p User Priority to Traffic Class Mappings, and Ingress QoS functionality. According to IEEE 802.1p, there are a maximum of eight traffic classes supported. This module can be configured to support fewer than eight traffic classes. The numerically highest traffic class is real-time traffic class, while all others are non-real-time traffic class. The real-time and non-real-time classification is performed by a combination of two subsystems: the VLAN Classifier module that provides the priority field in VLAN tag, and by the Ingress QoS - Priority Mapping Module that maps port number and VLAN priority to a traffic class. Real-time traffic is relayed promptly to the next module. Non-real-time traffic could be forwarded to the next module, buffered in a priority queue, or get dropped depending on the frame’s traffic class. If the frame is classified as realtime, or the traffic class for that frame does not

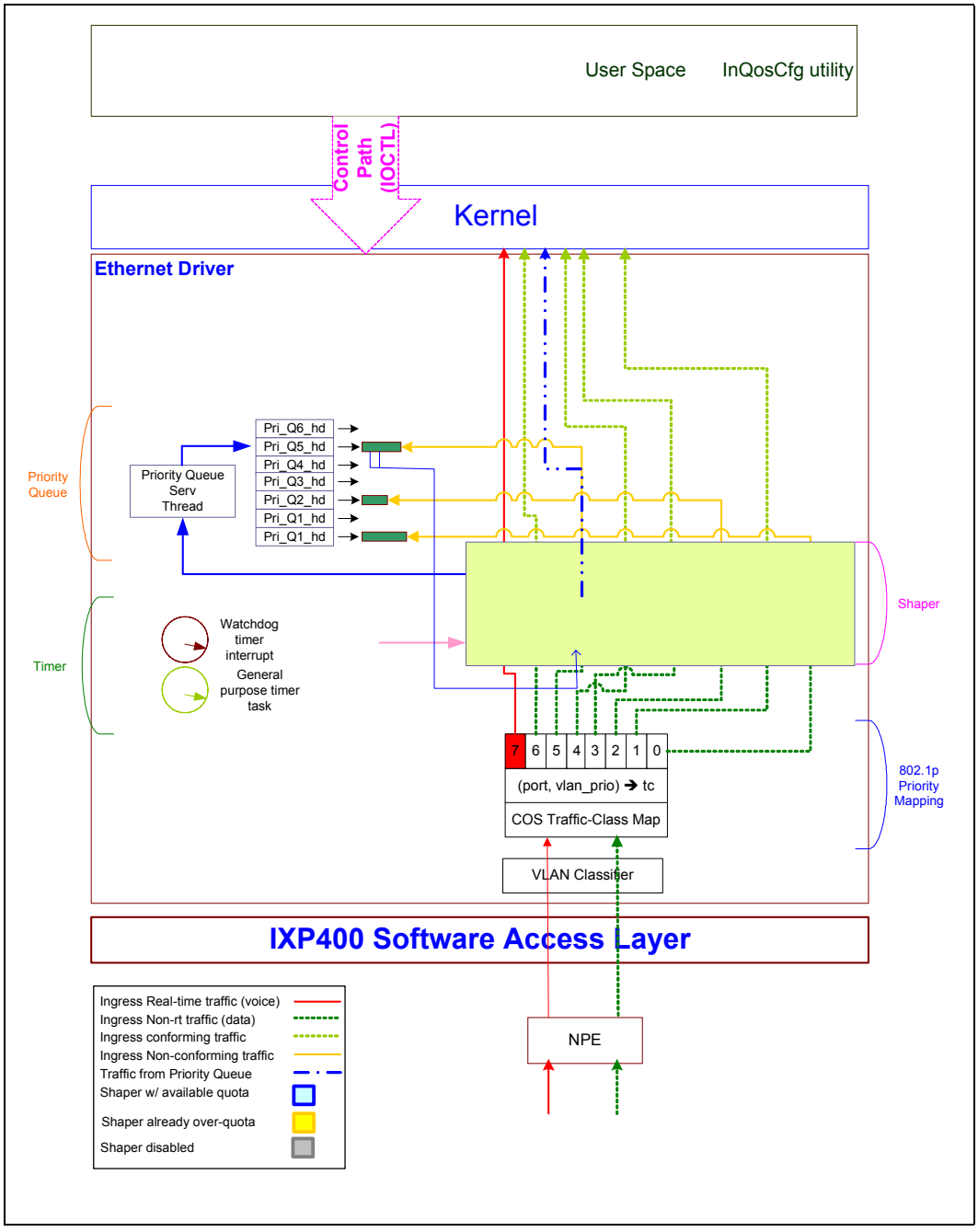
have shaper enabled, or "the shaper for that traffic class has available queues AND there are no frames buffered in the input queue for that traffic class", then that frame is forwarded to the next module.

Traffic shaping applies to those traffic classes that are not specified as “real-time”. For each traffic class, there are two types of shapers: Data Bytes shaper (D-type), and Frame Count shaper (F-type). There are two parameters associated with each shaper: Average Rate (avgD/avgF), and Peak Rate (peakD/peakF). The shaper is designed with the concept of a token bucket. The shaper design controls the long-term rate of traffic while also allowing some short-term bursts. The notation of the D-type shaper is used extensively in the network community, as it monitors the bandwidth of a certain traffic class. The purpose for an F-type shaper is to control the number of frames allowed for further processing. The F-type shaper is commonly used to limit the number of table lookups required by the host CPU, which is often a system bottleneck. The upper-layer user interface, however, can be configured to use both shapers simultaneously, one of the two shaper types at a time, or disable the shaper for some particular traffic class. When the shaper for a traffic class is disabled, frames belonging to that traffic class are passed to the next module directly.

When the shaper type (D-type, F-type, or both) is enabled for a traffic class, frames that are classified to this traffic class have to go through its corresponding shaper. When the corresponding shaper still has available quota and no frames are buffered in the priority queue, the frame is passed to the next module. Alternately, when there are frames buffered in the respective priority queue, a new frame is sent to the priority queue (regardless of the shaper status) as long as the priority queue is below its watermark threshold. If any of the enabled shapers is over-quota, then the frame is sent to priority queue for that traffic class. Before that frame is physically pushed to a priority queue, the water level of the priority queue corresponding to that traffic class is examined. If the water level is over-threshold (configurable), the frame is dropped instantly, otherwise the frame is pushed to the priority queue.

Periodically, the timer module is triggered and updates shapers that are enabled. After updating all enabled shapers, a signal is sent to the priority queue service routine, which pops frames from the high-priority traffic class queue to the low-priority traffic queue. For each frame at the head of a certain queue, the shaper status is checked before sending to the next module. If the shaper has available queues, the frame is popped and sent to the next module, after that the corresponding shapers are updated. If the shapers are already over-quota, then the priority queue service routine goes to the next priority queue for service.

**Figure 14. 802.1p User Priority to Traffic Class Mapping and Ingress QoS Modules – Component View**




Four software components are defined for 802.1p User Priority to Traffic Class Mappings and Ingress QoS. Those components are briefly described below and detailed in later sections. Traffic Shaper, Priority Mapping, and Ingress Queues deal with frame processing, while the Management Interface deals with configuration for each component and provides public APIs for other modules access.

**802.1p Priority Mapping and Ingress QoS Sub-Components**

- Traffic Shaper

  Ingress traffic rate control for kernel process.

- Priority Mapping

  Processes for IEEE 802.1Q/p User Priority to Traffic Class Mappings is handled in this component.

- Ingress Queues

  Up to eight priority queues for ingress frames.

- Management Interface

  Control path for maintaining associated database as well as shaper configuration.

The modularization of 802.1p User Priority to Traffic Class Mappings and Ingress QoS components (such as ingress queues and traffic shaper) makes it easy for updating/enhancing Ingress QoS functions. No effort is required for design changes in 802.1p User Priority to Traffic Class Mappings when a new queueing discipline is added into the ingress queue module, or when a new traffic control algorithm is defined for the traffic shaper.

## 4.1 Traffic Shaper Component

The purpose of the Traffic Shaper component is to control the rate of traffic sent to the next software module. In general, the highest traffic class is treated as real-time traffic. Real-time traffic is relayed promptly to the next software module. For non-real-time traffic, the Traffic Shaper determines if the frame should be passed to next module directly, queued in the ingress priority queues, or if the frame should be dropped. As previously stated, frames are sent to the priority queue under the condition that either traffic rate exceeds the shaper's configured rate or there are frames waiting in the corresponding priority queue. If rate limitation is not exceeded, and there are no frames buffered in the respective queue, frames are relayed to the next module. In the case where the frame must be delayed, the module first checks the water level of the particular priority queue. If the water level is above the high threshold, then the frame is discarded; otherwise frames are pushed to the corresponding priority queue.

If shapers for a traffic class are disabled, then traffic classified to this traffic class is treated as real-time traffic. Frames are passed to the next module directly.

Each time a frame is sent to the next software module, the shapers are updated accordingly. If both D-type and F-type shapers are enabled, then the quota for both types are updated. If only D-type is enabled, then only the quota for D-type is updated. The F-type shaper quota update process is the same.

Periodically, the timer (either timer task or interrupt) updates shaper parameters. The shaper parameters are:

- traffic class number: tc
- average data rate in byte-per-sec (BPS): aBPS
- average packet count in frame-per-sec (FPS): aFPS
- peak data rate in BPS: pBPS
- peak packet count per sec: pFPS
- type of shaper (D-type or F-type): type

The timer parameters are:

- type of timer: timer task or interrupt
- period of timer: in millisecond

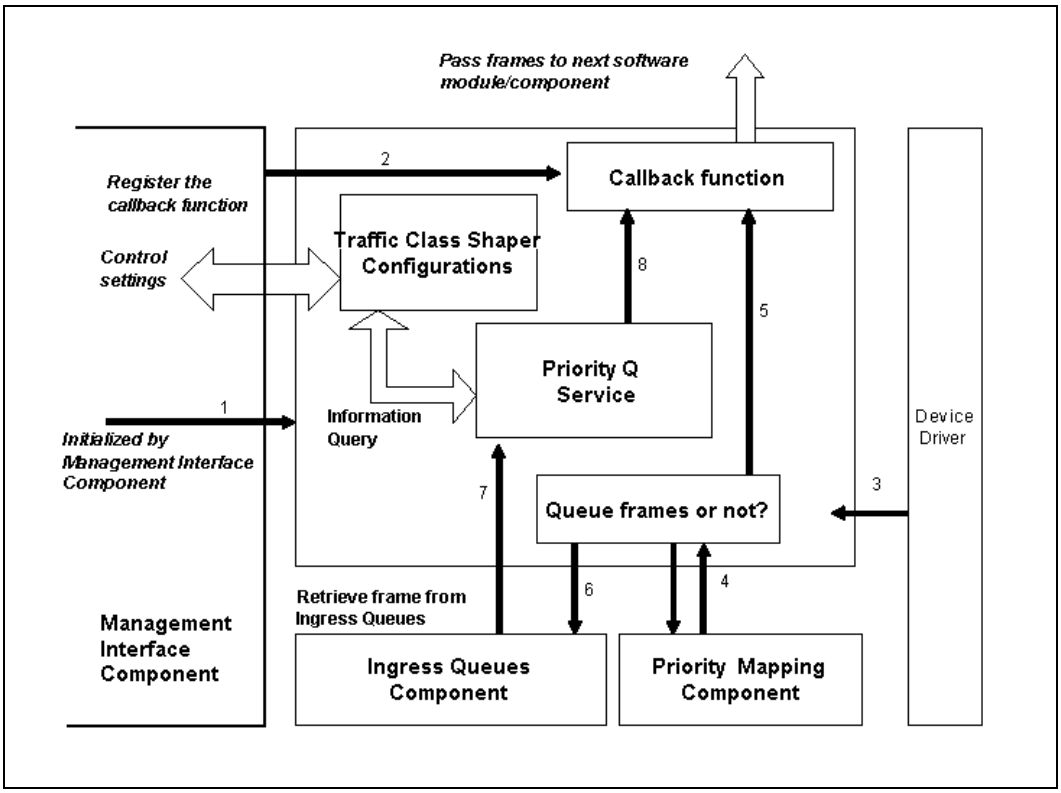
The functionality provided by the Traffic Shaper component includes:

- Initialize Traffic shaper and unload traffic shaper.
- Determine if the frame conforms the shaper configuration.
- Configure the traffic rate to next software module/component by
  - Number of frames
  - Number of bytes
- Provide/Reset statistical data on the number of serviced frames/conformed frames/frame buffered/frame dropped, etc.
- Register a callback function for the next software component.

### 4.1.1 External Interactions and Dependencies

This software component is initialized and configured by the Management Interface component. The Management Interface component is required to register a proper callback function for the Traffic Shaper component to enable passing frames to the next software module/component. The Traffic Shaper component utilizes services of Ingress Queues to retrieve frames from its queues and to push frames from Ingress Queues to Priority Queues. Interactions and dependencies are illustrated in Figure 15.

**Figure 15. Traffic Shaper Component Interactions and Dependencies**




1. The device driver (via Management Interface) calls the API (ixQosIngressShaperInit) to initialize the Traffic Shaper component.
2. The device driver calls the API (ixQoSIngressShaperCallbackRegister) to register the callback function.
3. The device driver calls the API (ixQosIngressProcess) to initiate the Ingress QoS process.
4. Traffic Shaper calls the API (ixQos1pPriority2TCFrameMap) to determine if frames are real-time. If frames are real-time, or non real-time traffic with traffic rate not exceeding the configured rate and no frames in the corresponding queue, execute next step. Otherwise, execute step 6.
5. Call the registered callback function to relay frames to next software module.
6. Call the API (ixQosIngressQAllowPush) to check water level. If the water level is below high water mark, drop the packet. Otherwise, go to step 7.
7. Call the API (ixQosIngressQPush) of Ingress Queue component to queue the frame.
8. Priority Queue Service routine is executed by scheduler and de-queue frames from queues.

## 4.1.2 Key Assumptions

- "IXE_DRV_IN_QOS_DEBUG" definition is used to determine whether or not the code is operating under debugging environment. If "IXE_DRV_IN_QOS_DEBUG" is not defined, all input parameters to this component should be valid. For performance considerations, under normal operation no routines in this component perform any input parameter checking.

## 4.2 Priority Mapping Component

This component maps the user priority (0~ 7) (determined in VLAN classification) of a received frame into the corresponding traffic class value. Frames are classified into different classes and call the services of the Traffic Shaper component to determine if frames should be queued or not.

**Figure 16. 802.1p User Priority to Traffic Class Mapping**

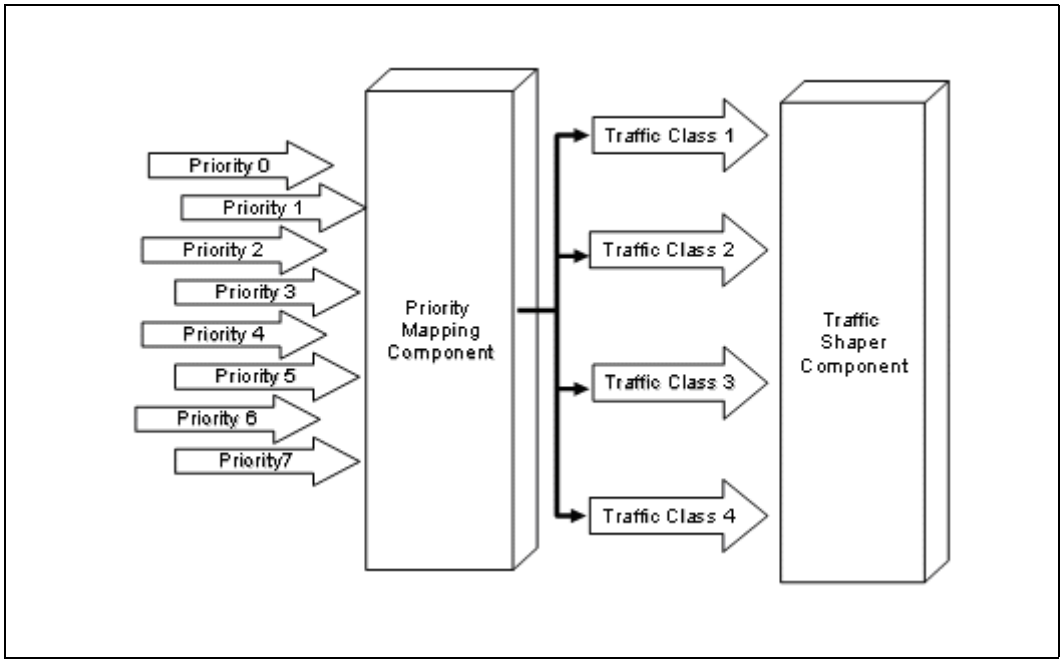



**Table 2. User Priority to Traffic Class Defaults and Recommendations**

| User Priority | Traffic Class (Default) | Traffic Class (Recommended for 2 Traffic Class Queues) | Traffic Class (Recommended for 4 Traffic Class Queues) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 |
| 4 | 0 | 0 | 1 |
| 5 | 0 | 0 | 2 |
| 6 | 0 | 0 | 2 |
| 7 | 0 | 1 (highest priority) | 3 (highest priority) |

The functionality provided by the Priority Mapping component includes:

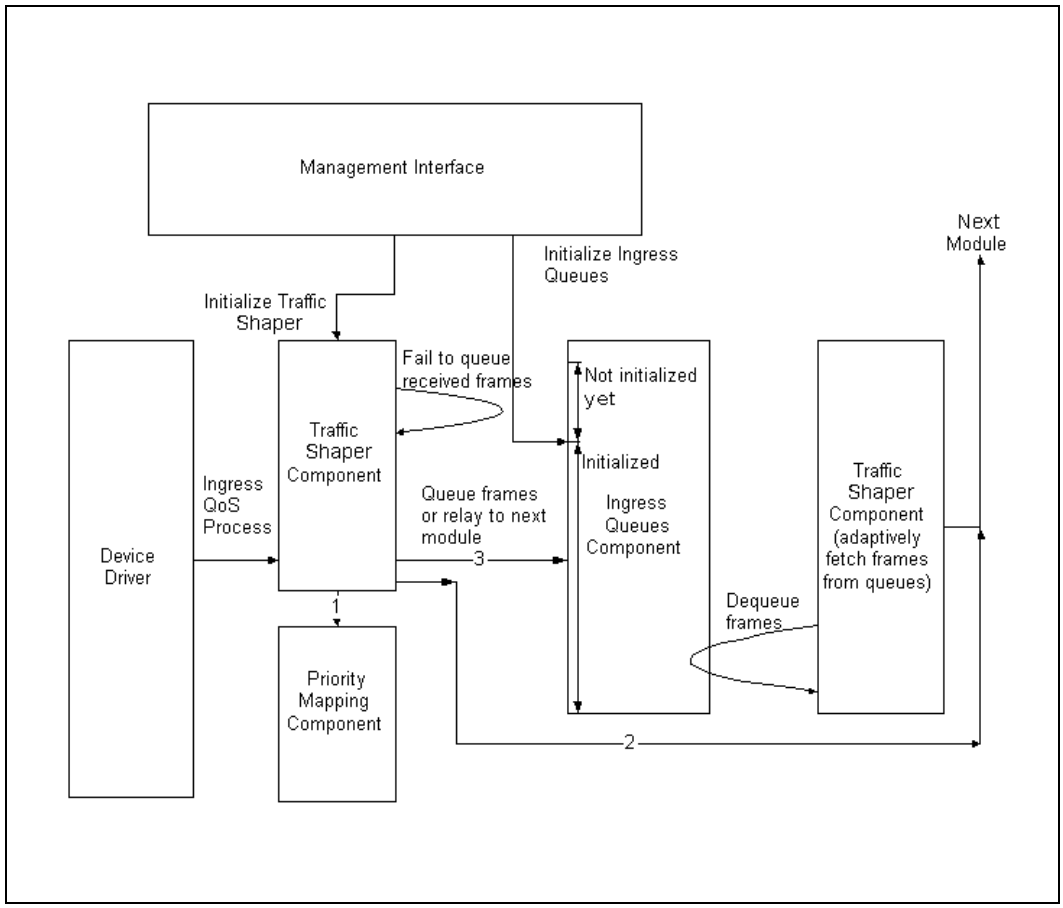
- Initialize and unload Priority Mapping component.
- Support 802.1Q/p User Priority to Traffic Class Mappings of received frames.

- Interface to configure values of User Priority to Traffic Class Mappings.
- Call services of Traffic Shaper component to determine if received frames are queued or not.

### 4.2.1 External Interactions and Dependencies

This software component is initialized and configured by the Management Interface component. The device driver utilizes services of this component for Ingress QoS data processing. The Traffic Shaper component uses the Priority Mapping component to determine the traffic class of received frames.

**Figure 17. Priority Mapping Interactions and Dependencies**




1. The Traffic Shaper component calls the API (ixQos1pPriority2TCFrameMap) to determine the traffic class of received frames. If it is real-time traffic or non real-time but decide to relay to next module/component promptly, execute step 2. Otherwise execute step 3.
2. Traffic Shaper calls the registered callback function to relay frames to the next software module.
3. Traffic Shaper calls the API (ixQosIngressQPush) of Ingress Queue component to queue the received frames into related priority queues.

### 4.2.2 Key Assumptions

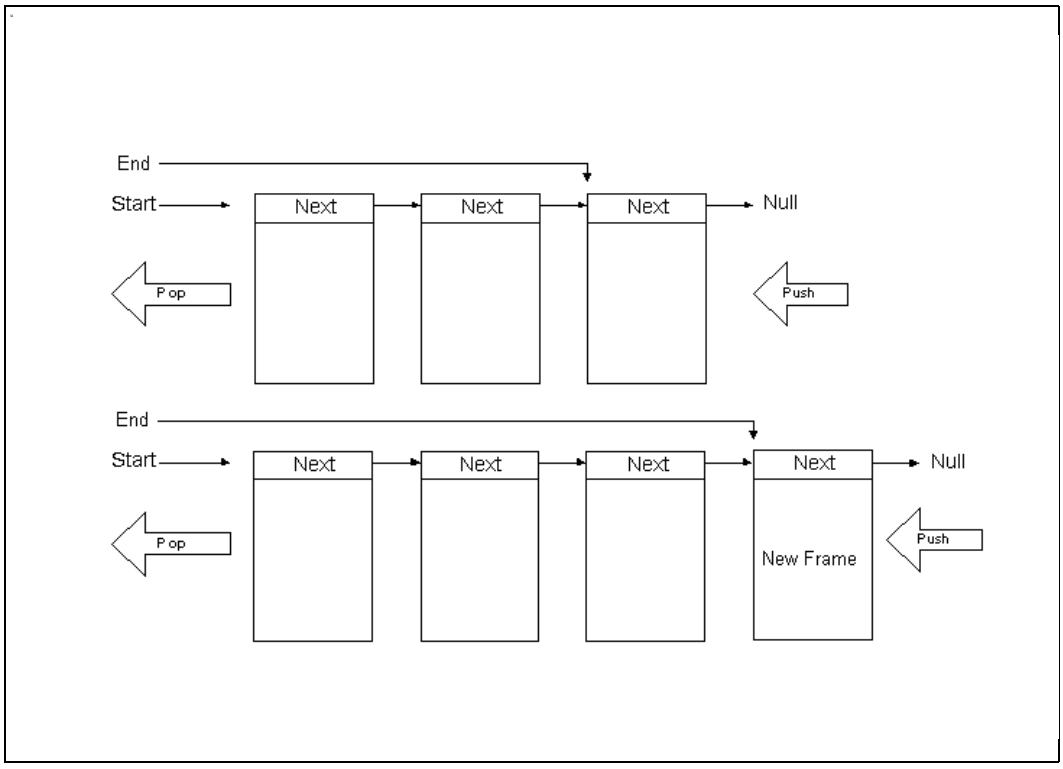
- "IXE_DRV_IN_QOS_DEBUG" definition is used to determine whether or not the code is operating under a debugging environment. If "IXE_DRV_IN_QOS_DEBUG" is not defined, all input parameters to this component should be valid. For performance considerations, under normal operation no routines in this component perform any input parameter checking.
- The Number of Traffic Classes supported in this component and the Number of Queues supported in the Ingress Queues component are identical.

## 4.3 Ingress Queues

As per the IEEE 802.1Q/p standard, the "Forwarding Process" may provide more than one transmission queue for a given port. Frames are assigned to storage queue(s) on the basis of their user priority using a traffic class table. This component simulates these queues in the ingress path. There are two advantages from these ingress queues:

- It is compliant with the 802.1Q/p standard.
- By queueing different priority frames into different queues, more computing power of the host processor is reserved to service frames in a higher priority queue.

**Figure 18. Ingress Queues**




The functionality provided by the Ingress Queues component is:

- Initialize and unload Ingress Queue components
- Support up to eight queues

- Push a frame to the specific queue
- Pop a frame from the specific queue
- Obtain information for a specific queue

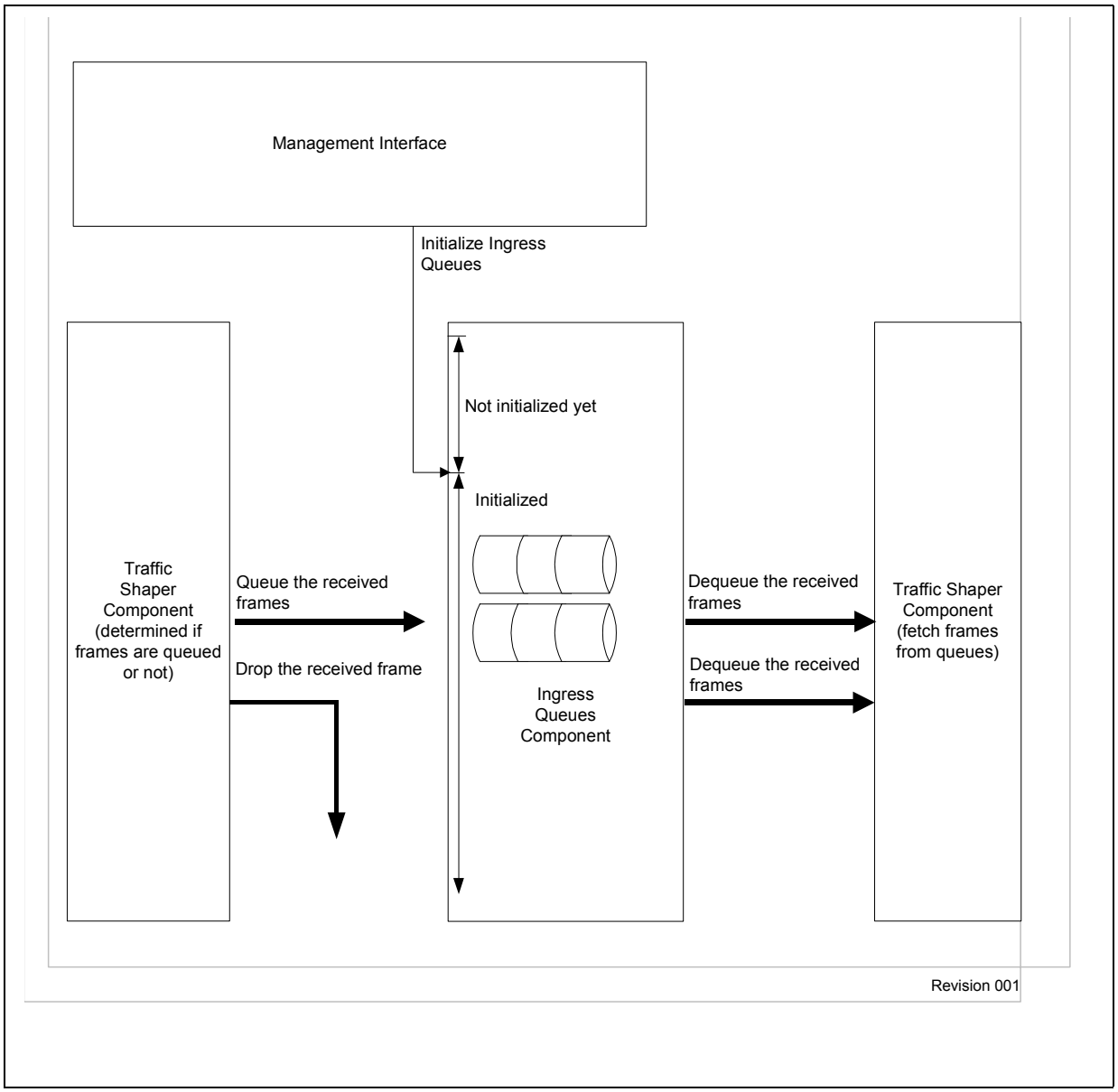
### 4.3.1 External Interactions and Dependencies

This software component is initialized and configured by Management Interface component. Frames are pushed into (and popped from) queues by the Traffic Shaper component.

### 4.3.2 Key Assumptions

- "IXE_DRV_IN_QOS_DEBUG" definition is used to determine whether or not the code is operating under a debugging environment. If "IXE_DRV_IN_QOS_DEBUG" is not defined, all input parameters to this component should be valid. For performance considerations, under normal operation no routines in this component perform any input parameter checking.
- The Number of Traffic Classes in this component and the Number of Queues in Ingress Queues component are identical.

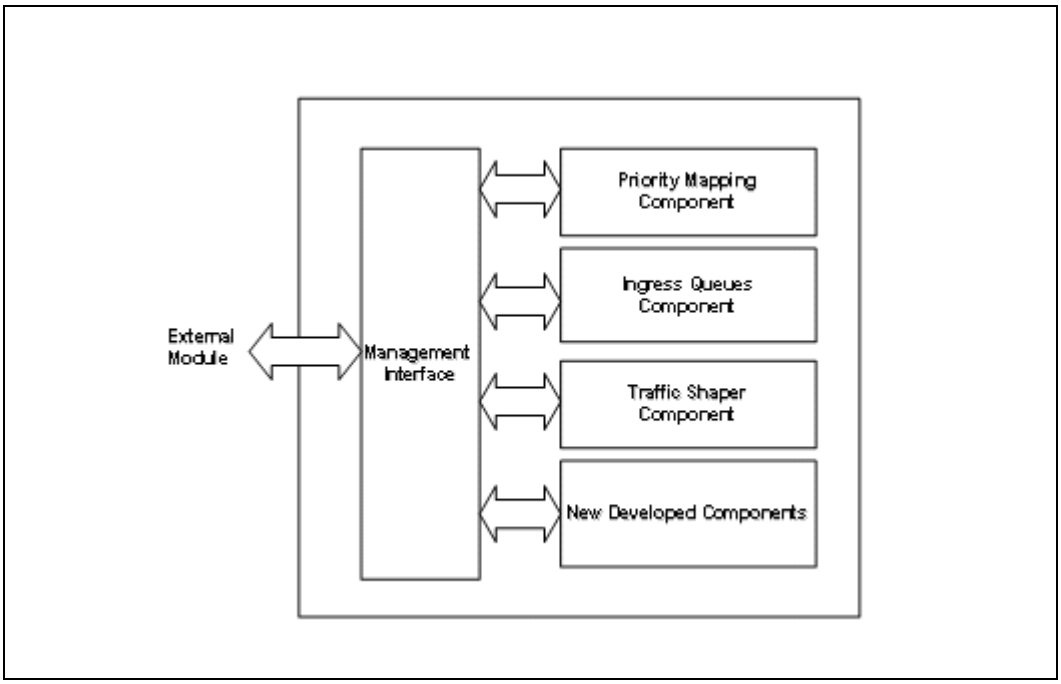
**Figure 19. Dependencies and Interactions for Ingress Queues Component**




## 4.4 Management Interface Component

This component provides the public interface (i.e., control path) for external modules to configure the behavior of 802.1p User Priority to Traffic Class Mappings and Ingress QoS module. The IOCTL parser in Ethernet device driver should utilize this interface to access services of the QoS module. Direct access to services (or APIs) in other components of this module is not supported.

**Figure 20. Interactions of the QoS Module Management Interface Sub-Component**



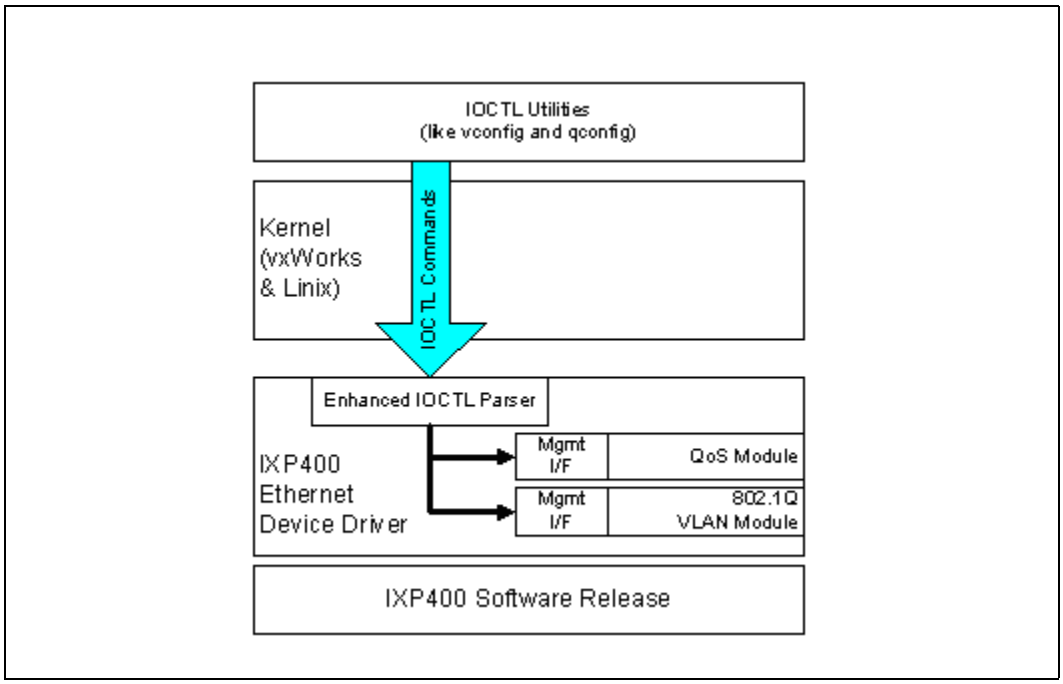

### 4.4.1 Key Dependencies

Regardless of whether or not “IXE_DRV_IN_QOS_DEBUG” is defined, all routines in this component perform necessary input parameter checking.

## 5.0 IOCTL Enhancements for Ethernet Drivers

The purpose of this software is to extend existing IOCTL functionality in the IXP400 Ethernet device drivers. New IOCTL commands are defined to support new features for 802.1Q VLAN and QoS Modules. These commands are grouped into a pair of IOCTL utilities, **vconf** and **qconf**, which are used to access VLAN and QoS services in the IXP400 Ethernet device driver. The modified IOCTL Parser recognizes these IOCTL commands and in turn executes associated services in the VLAN and QoS modules.

**Figure 21. System View of IOCTL Utilities and Parser**




The **vconf** utility sends management API calls via IOCTL commands to the VLAN Module, while **qconf** does the same for the QoS Module. More detailed information regarding the syntax of these utilities is available in the Intel® IXP400 Software: VLAN and QoS Application Version 1.0 *Release Notes*.

# 6.0 API Reference

This section contains the VLAN and QoS Application v1.0 APIs and data structures.

## API Index (Sheet 1 of 3)

## API Index (Sheet 2 of 3)

### API Index (Sheet 3 of 3)



# 6.1 VLAN Module

## 6.1.1 VLAN Module Interface

| **Prototype:** | IX_STATUS **ixVlanModuleInitialize** (void); | |
|---|---|---|
| **Parameters:**<br>n/a | **Description:** | **I/O:** |
| **Return:** | IX_SUCCESS | |
| **Description:** | This function is used to startup the VLAN module, including all VLAN databases. No parameter is required for this function. | |

| Prototype: | IX_STATUS **ixVlanModuleEnableStatusSet** ( BOOL ***status***); | |
|---|---|---|
| **Parameters:**<br>status | **Description:**<br>VLAN module is to be enabled or disabled according to its value, TRUE or FALSE respectively. | **I/O:**<br>I |
| **Return:** | IX_SUCCESS | |
| **Description:** | This function is used to enable/disable the VLAN module. | |

| Prototype: | IX_STATUS **ixVlanModuleEnableStatusGet** ( BOOL ****status***); | |
|---|---|---|
| **Parameters:**<br>*status | **Description:**<br>VLAN module status is enabled or disabled according to its value, TRUE or FALSE respectively. | **I/O:**<br>O |
| **Return:** | IX_FAIL<br>IX_SUCCESS | |
| **Description:** | This function is used to get the status of enabling of VLAN module. | |

## 6.1.2 VLAN Database Control Interface

| Prototype: | IX_STATUS **ixVlanDBVlanCreate** ( VLAN_ID ***vid***); | |
|---|---|---|
| **Parameters:**<br>vid | **Description:**<br>The VLAN of vid 1 is always created whenever the VLAN module is enabled. The legal range of "vid" is from 1 to 4094. | **I/O:**<br>I |
| **Return:** | IX_FAIL – Illegal VLAN id, VLAN already exists; or no VLAN entry available<br>IX_SUCCESS | |
| **Description:** | This function is used to create a VLAN. | |

| Prototype: | IX_STATUS **ixVlanDBVlanDestroy** ( VLAN_ID ***vid***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| vid | The VLAN of vid 1 is always created whenever the VLAN module is enabled. The legal range of "vid" is from 1 to 4094. | I |
| **Return:** | IX_FAIL – Illegal VLAN id, VLAN does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to delete an existing VLAN. | |

| Prototype: | IX_STATUS **ixVlanDBMembershipSet** ( VLAN_ID ***vid***, PORT_BITMAP ***member_bmp*** PORT_BITMAP ***egress_bmp***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| vid | The VLAN of vid 1 is always created whenever the VLAN module is enabled. The legal range of "vid" is from 1 to 4094. | I |
| member_bmp | Specifies the member ports to add to the VLAN. | I |
| egress_bmp | Set tagged or untagged egress status. | I |
| **Return:** | IX_FAIL – VLAN does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to join ports as members of a VLAN. This function is also used to define the egress type of each port. | |

| **Prototype:** | IX_STATUS **ixVlanDBMembershipGet** ( VLAN_ID ***vid***, PORT_BITMAP ****member_bmp*** PORT_BITMAP ****egress_bmp***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| vid | The VLAN of vid 1 is always created whenever the VLAN module is enabled. The legal range of "vid" is from 1 to 4094. | I |
| *member_bmp | Receives the member ports to add to the VLAN. | O |
| *egress_bmp | Tagged or untagged egress status. | O |
| **Return:** | IX_FAIL – VLAN does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to get the membership ports of a VLAN, and the egress type of these members | |

| **Prototype:** | IX_STATUS **ixVlanDBMemberSet** ( VLAN_ID ***vid***, PORT_ID ***pid***, BOOL ***is_member***, BOOL, ***is_untagged_egress*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| vid | The VLAN of vid 1 is always created whenever the VLAN module is enabled. The legal range of "vid" is from 1 to 4094. | I |
| pid | Specifies the port | I |
| is_member | TRUE == add port to VLAN, FALSE == remove port from VLAN | I |
| is_untagged_egress | Tagged or untagged egress status TRUE == tagged, FALSE ==- untagged. | I |
| **Return:** | IX_FAIL – Illegal Port ID, port has PVID for this VLAN, VLAN does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to add or remove ports to or from an existing VLAN | |

| Prototype: | IX_STATUS **ixVlanDBMemberGet** ( VLAN_ID ***vid***, PORT_ID ***pid***, BOOL ****is_member***, BOOL, ****is_untagged_egress*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| vid | The VLAN of vid 1 is always created whenever the VLAN module is enabled. The legal range of "vid" is from 1 to 4094. | I |
| pid | Specifies the port | I |
| *is_member | TRUE == add port to VLAN, FALSE == remove port from VLAN | O |
| *is_untagged_egress | Tagged or untagged egress status TRUE == tagged, FALSE ==- untagged. | O |
| **Return:** | IX_FAIL – Illegal Port ID.<br>IX_SUCCESS | |
| **Description:** | This function is used to determine whether a port is the member of a particular VLAN. | |

| Prototype: | IX_STATUS **ixVlanDBFirstVlanIdGet** ( VLAN_ID ****vid*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *vid | The VLAN of vid 1 is always created whenever the VLAN module is enabled. The legal range of "vid" is from 1 to 4094. | O |
| **Return:** | IX_FAIL – No VLAN exists.<br>IX_SUCCESS | |
| **Description:** | This function provides the VLAN id of the first VLAN in the database. | |

| Prototype: | IX_STATUS **ixVlanDBNextVlanIdGet** ( VLAN_ID ****vid*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *vid | The VLAN of vid 1 is always created whenever the VLAN module is enabled. The legal range of "vid" is from 1 to 4094. | O |
| **Return:** | IX_FAIL – No additional VLANs exist.<br>IX_SUCCESS | |
| **Description:** | This function provides the VLAN ids of all VLANs in the database. On each call, it returns the next vid in the VLAN database. If there are no additional VLANs, it returns IX_FAIL. It is OK to call this function without first calling ixVlanDbFirstVlanIdGet. | |

## 6.1.3 Port Database Control Interface

| Prototype: | IX_STATUS **ixVlanDBPortAcceptFrameTypeSet** ( PORT_ID ***pid***, ACCEPT_TYPE ***type*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| pid | Specifies the port | I |
| type | Specifies the acceptable frame type,0 == ALL_FRAME, 1 == ONLY_TAGGED_FRAME. | I |
| **Return:** | IX_FAIL – Illegal Port.<br>IX_SUCCESS | |
| **Description:** | This function is used to set the acceptable frame type of a port. | |

| Prototype: | IX_STATUS **ixVlanDBPortAcceptFrameTypeGet** ( PORT_ID ***pid***, ACCEPT_TYPE ****type*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| pid | Specifies the port | I |
| type | Specifies the acceptable frame type,0 == ALL_FRAME, 1 == ONLY_TAGGED_FRAME. | O |
| **Return:** | IX_FAIL – Illegal Port.<br>IX_SUCCESS | |
| **Description:** | This function is used to determine whether a port can accept VLAN-tagged frames. | |

| Prototype: | IX_STATUS **ixVlanDBPortIngressFilterStatusSet** ( PORT_ID ***pid***, BOOL ***status***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| pid | Specifies the port | I |
| status | Specifies ingress filtering,0 == enabled, 1 == disabled. | I |
| **Return:** | IX_FAIL – Illegal Port.<br>IX_SUCCESS | |
| **Description:** | This function is used to set whether a port should use ingress filtering. | |

| Prototype: | IX_STATUS **ixVlanDBPortIngressFilterStatusGet** ( PORT_ID ***pid***, BOOL ****status***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| pid | Specifies the port | I |
| *status | Specifies ingress filtering,0 == enabled, 1 == disabled. | O |
| **Return:** | IX_FAIL – Illegal Port.<br>IX_SUCCESS | |
| **Description:** | This function is used to determine whether a port uses ingress filtering. | |

| Prototype: | IX_STATUS **ixVlanDBPortRuleSet** ( PORT_ID ***pid***, VLAN_ID ***vid***, PRIORITY ***priority*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| pid | Specifies the port | I |
| vid | Default VLAN ID of the port | I |
| priority | 802.1q priority level. | I |
| **Return:** | IX_FAIL – Illegal port or VLAN does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to set the default VLAN id (PVID) and priority for a port. | |

| Prototype: | IX_STATUS **ixVlanDBPortRuleGet** ( PORT_ID ***pid***, VLAN_ID ****vid***, PRIORITY ****priority*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| pid | Specifies the port | I |
| *vid | Default VLAN ID of the port | O |
| *priority | 802.1q priority level. | O |
| **Return:** | IX_FAIL – Illegal Port.<br>IX_SUCCESS | |
| **Description:** | This function is used to determine the default VLAN id (PVID) and priority for a port. | |

| Prototype: | IX_STATUS **ixVlanDBPortRuleReset** ( PORT_ID ***pid*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| pid | Specifies the port | I |
| **Return:** | IX_FAIL – Illegal Port.<br>IX_SUCCESS | |
| **Description:** | This function is used to reset the default VLAN id (PVID), priority, ingress filter status and frame type accept status. | |

## 6.1.4 MAC Rule Database Control Interface

| Prototype: | IX_STATUS **ixVlanDBMacRuleAdd** ( MAC_RULE ****mac_rule***, RULE_ID ****rid*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *mac_rule | Structure containing the MAC-based rule. | I |
| *rid | Rule ID number. | O |
| **Return:** | IX_FAIL – Illegal rule contents (illegal priority, illegal VLAN), rule pool is full.<br>IX_SUCCESS | |
| **Description:** | This function is used to add a new MAC-based rule to the VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanDBMacRuleDelete** (RULE_ID ***rid*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| **Return:** | IX_FAIL<br>IX_SUCCESS | |
| **Description:** | This function is remove an existing MAC-based rule from the VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanDBMacRuleGet** ( RULE_ID ***rid***, MAC_RULE ****mac_rule*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| *mac_rule | Structure containing the MAC-based rule. | O |
| **Return:** | IX_FAIL – Rule ID does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to read the contents of an existing MAC-based rule to the VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanDBMacRuleFind** ( MAC_RULE ****mac_rule***, RULE_ID ****rid*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *mac_rule | Structure containing the MAC-based rule to match. | I |
| *rid | Rule ID number. | O |
| **Return:** | IX_FAIL – No matching rule found.<br>IX_SUCCESS | |
| **Description:** | This function is used to get an existing MAC-based rule ID from the VLAN classifier. The seach will attempt to find an existing rule that matches to content of *mac_rule. | |

| Prototype: | IX_STATUS **ixVlanDBMacRuleActivateStatusSet** ( RULE_ID ***rid***, BOOL ***activate***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| activate | Specifies if rule is active,0 == disabled, 1 == active. | I |
| **Return:** | IX_FAIL — Rule ID does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to activate an existing MAC-based rule ID in the VLAN classifier. Entries can be activated even when the MAC-based classifier is disabled. (ixVlanClassMacClassifierStatusSet). Those entries will then be used by the classified once the classifier is enabled. | |

| | | |
|---|---|---|
| **Prototype:** | IX_STATUS **ixVlanDBMacRuleActivateStatusGet** ( RULE_ID ***rid***, BOOL ***activate***); | |
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| *activate | Specifies if rule is active,0 == disabled, 1 == active. | O |
| **Return:** | IX_FAIL — Rule ID does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to detect whether an existing MAC-based rule ID in the VLAN classifier is active. | |

| | | |
|---|---|---|
| **Prototype:** | IX_STATUS **ixVlanDBFirstMacRuleIdGet** ( RULE_ID ****rid*** ); | |
| **Parameters:** | **Description:** | **I/O:** |
| *rid | Rule ID number. | O |
| **Return:** | IX_FAIL – No MAC-based rules exist.<br>IX_SUCCESS | |
| **Description:** | This function provides the rule id of the first MAC-based rule in the VLAN classifier. | |

| | | |
|---|---|---|
| **Prototype:** | IX_STATUS **ixVlanDBNextMacRuleIdGet** ( RULE_ID ****rid*** ); | |
| **Parameters:** | **Description:** | **I/O:** |
| *rid | Rule ID number. | O |
| **Return:** | IX_FAIL – No MAC-based rules exist.<br>IX_SUCCESS | |
| **Description:** | This function provides the rule id of all MAC-based rules in the VLAN classifier. On each call, it returns the next MAC-rule in the VLAN database. If there are no additional rules, it returns IX_FAIL. It is OK to call this function without first calling ixVlanDbFirstMacRuleIdGet. | |

| Prototype: | IX_STATUS **ixVlanDBMacRuleGroupChange** ( RULE_ID ***rid***, GROUP_ID ***gid***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| gid | Group ID number for the rule. Default value = 8, legal values 0 to 15. | I |
| **Return:** | IX_FAIL – Illegal Rule ID.<br>IX_SUCCESS | |
| **Description:** | This function is used to activate an existing MAC-based rule ID in the VLAN classifier. A group is a collection of MAC or Protocol-based rules. A group with a higher gid is searched before lower-gid-value groups. The classified stops rule searching for a frame as soon as the first rule match is made. | |

| Prototype: | IX_STATUS **ixVlanDBMacRuleGroupGet** ( RULE_ID ***rid***, GROUP_ID ****gid***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| *gid | Group ID number for the rule. Default value = 8, legal values 0 to 15. | O |
| **Return:** | IX_FAIL – Illegal Rule ID.<br>IX_SUCCESS | |
| **Description:** | This function is used to determine which group an existing MAC-based rule belongs to in the VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanDBMacRuleHitGet** ( RULE_ID ***rid***, unsigned long ****hit***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| *hit | Number of packets. | O |
| **Return:** | IX_FAIL – Illegal Rule ID.<br>IX_SUCCESS | |
| **Description:** | This function is used to retrieve the number of frames that have matched the specified MAC-based rule by the VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanDBMacRuleResetAll** (); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| **Return:** | IX_SUCCESS | |
| **Description:** | This function is used to remove all MAC-based rules from the VLAN classifier. | |

## 6.1.5 Protocol Rule Database Control Interface

| Prototype: | IX_STATUS **ixVlanDBProtocolRuleAdd** ( IP_RULE ***ip_rule***, RULE_ID ***rid*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *ip_rule | Structure containing the Protocol-based rule. | I |
| *rid | Rule ID number. | O |
| **Return:** | IX_FAIL – Illegal rule contents (illegal priority, illegal VLAN), rule pool is full.<br>IX_SUCCESS | |
| **Description:** | This function is used to add a new Protocol-based rule to the VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanDBProtocolRuleDelete** (RULE_ID ***rid*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| **Return:** | IX_FAIL — Rule ID does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is remove an existing Protocol-based rule from the VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanDBProtocolRuleGet** ( RULE_ID ***rid***, IP_RULE ****ip_rule*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| *ip_rule | Structure containing the Protocol-based rule. | O |
| **Return:** | IX_FAIL – Rule ID does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to read the contents of an existing Protocol-based rule to the VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanDBProtocolRuleFind** ( IP_RULE ****ip_rule***, RULE_ID ****rid*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *ip_rule | Structure containing the Protocol-based rule to match. | I |
| *rid | Rule ID number. | O |
| **Return:** | IX_FAIL – No matching rule found.<br>IX_SUCCESS | |
| **Description:** | This function is used to get an existing Protocol-based rule ID from the VLAN classifier. The seach will attempt to find an existing rule that matches to content of *ip_rule. | |

| Prototype: | IX_STATUS **ixVlanDBProtocolRuleActivateStatusSet** ( RULE_ID ***rid***, BOOL ***activate***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| activate | Specifies if rule is active,0 == disabled, 1 == active. | I |
| **Return:** | IX_FAIL — Rule ID does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to activate an existing Protocol-based rule ID in the VLAN classifier. Entries can be activated even when the Protocol-based classifier is disabled. (ixVlanClassProtocolClassifierStatusSet). Once the classifier is enabled, those entries will then be used by the classifier. | |

| | | |
|---|---|---|
| **Prototype:** | IX_STATUS **ixVlanDBProtocolRuleActivateStatusGet** ( RULE_ID ***rid***, BOOL ****activate***); | |
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| *activate | Specifies if rule is active,0 == disabled, 1 == active. | O |
| **Return:** | IX_FAIL — Rule ID does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to detect whether an existing Protocol-based rule ID in the VLAN classifier is active. | |

| | | |
|---|---|---|
| **Prototype:** | IX_STATUS **ixVlanDBFirstProtocolRuleIdGet** ( RULE_ID ****rid*** ); | |
| **Parameters:** | **Description:** | **I/O:** |
| *rid | Rule ID number. | O |
| **Return:** | IX_FAIL – No Protocol-based rules exist.<br>IX_SUCCESS | |
| **Description:** | This function provides the rule id of the first Protocol-based rule in the VLAN classifier. | |

| | | |
|---|---|---|
| **Prototype:** | IX_STATUS **ixVlanDBNextProtocolRuleIdGet** ( RULE_ID ****rid*** ); | |
| **Parameters:** | **Description:** | **I/O:** |
| *rid | Rule ID number. | O |
| **Return:** | IX_FAIL – No Protocol-based rules exist.<br>IX_SUCCESS | |
| **Description:** | This function provides the rule id of all Protocol-based rules in the VLAN classifier. On each call, it returns the next Protocol-rule in the VLAN database. If there are no additional rules, it returns IX_FAIL. It is OK to call this function without first calling ixVlanDbFirstProtocolRuleIdGet. | |

| Prototype: | IX_STATUS **ixVlanDBProtocolRuleGroupChange** ( RULE_ID ***rid***, GROUP_ID ***gid***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| gid | Group ID number for the rule. Default value = 8, legal values 0 to 15. | I |
| **Return:** | IX_FAIL — Rule ID does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to activate an existing Protocol-based rule ID in the VLAN classifier. A group is a collection of MAC or Protocol-based rules. A group with a higher gid is searched before lower-gid-value groups. The classified stops rule searching for a frame as soon as the first rule match is made. | |

| Prototype: | IX_STATUS **ixVlanDBProtocolRuleGroupGet** ( RULE_ID ***rid***, GROUP_ID ****gid***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| *gid | Group ID number for the rule. Default value = 8, legal values 0 to 15. | O |
| **Return:** | IX_FAIL — Rule ID does not exist.<br>IX_SUCCESS | |
| **Description:** | This function is used to determine which group an existing Protocol-based rule belongs to in the VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanDBProtocolRuleHitGet** ( RULE_ID ***rid***, unsigned long ****hit***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| rid | Rule ID number. | I |
| *hit | Number of packets. | O |
| **Return:** | IX_FAIL – Illegal Rule ID.<br>IX_SUCCESS | |
| **Description:** | This function is used to retrieve the number of frames that have matched the specified Protocol-based rule by the VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanDBProtocolRuleResetAll** (); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| **Return:** | IX_SUCCESS | |
| **Description:** | This function is used to remove all Protocol-based rules from the VLAN classifier. | |

## 6.1.6 VLAN Classifier Control Interface

| Prototype: | IX_STATUS **ixVlanClassMacClassifierStatusSet** ( BOOL ***status***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| status | The status is enabled or disabled according to its value, TRUE or FALSE, respectively. | I |
| **Return:** | IX_SUCCESS | |
| **Description:** | This function is used to set the status of the MAC-based VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanClassMacClassifierStatusGet** ( BOOL ****status***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *status | The status is enabled or disabled according to its value, TRUE or FALSE, respectively. | O |
| **Return:** | IX_FAIL — Status == NULL.<br>IX_SUCCESS | |
| **Description:** | This function is used to get the status of the MAC-based VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanClassProtocolClassifierStatusSet** ( BOOL ***status***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| status | The status is enabled or disabled according to its value, TRUE or FALSE, respectively. | I |
| **Return:** | IX_FAIL<br>IX_SUCCESS | |
| **Description:** | This function is used to set the status of the Protocol-based VLAN classifier. | |

| Prototype: | IX_STATUS **ixVlanClassProtocolClassifierStatusGet** ( BOOL ****status***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *status | The status is enabled or disabled according to its value, TRUE or FALSE, respectively. | O |
| **Return:** | IX_FAIL<br>IX_SUCCESS | |
| **Description:** | This function is used to get the status of the Protocol-based VLAN classifier. | |

## 6.1.7 VLAN Module Data Path

| Prototype: | IX_STATUS **ixVlanClassPacketClassify** ( IX_MBUF ***pMblk***, PORT_ID ***pid***, DIRECTION ***direction***, VLAN_ID ***vid***, PRIORITY ***priority*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *pMblk | The mbuf for that frame. IX_MBUF for VxWorks*, sk_buff for Linux*. | IO |
| pid | The port where the frame is received or send, depending on direction. | I |
| direction | INGRESS or EGRESS | I |
| *vid | VLAN ID or Port VLAN ID (PVID) if no VID. | O |
| *priority | User Priority tag | O |
| **Return:** | IX_FAIL – Illegally tagged frame.<br>IX_SUCCESS – The frame was successfully classified. | |
| **Description:** | This function is used to classify a frame/packet according to any applicable rules in the VLAN classification database. | |

| Prototype: | IX_STATUS **ixVlanClassPacketClassify** ( IX_MBUF ***pMblk***, PORT_ID ***pid***, DIRECTION ***direction***, VLAN_ID ***vid***, PRIORITY ***priority*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *pMblk | The mbuf for that frame. IX_MBUF for VxWorks, sk_buff for Linux. | IO |
| pid | The port where the frame is received or send, depending on direction. | I |
| direction | INGRESS or EGRESS | I |
| *vid | VLAN ID or Port VLAN ID (PVID) if no VID. | O |
| *priority | User Priority tag | O |
| **Return:** | IX_FAIL – Illegally tagged frame.<br>IX_SUCCESS – The frame was successfully classified. | |
| **Description:** | This function is used to classify a frame/packet according to any applicable rules in the VLAN classification database. | |

| Prototype: | IX_STATUS **ixVlanIngressProcess** ( PORT_ID ***pid***, IX_MBUF ****pMblk*** , BOOL ****discard***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| pid | The port where the frame is received. | I |
| *pMblk | The mbuf for that frame. IX_MBUF for VxWorks, sk_buff for Linux. | I |
| *discard | TRUE if VLAN module makes decision to drop this frame. | O |
| **Return:** | IX_FAIL – Fails frame filter on ingress port, not member of Ingress VLAN, Illegally tagged frame.<br>IX_SUCCESS – The upper level should be notified that a new frame is received. | |
| **Description:** | This function is used to receive a frame/packet, and check it against applicable filters, VLAN membership, and submit to the classifier. | |

| Prototype: | IX_STATUS **ixVlanEgressProcess** ( PORT_ID ***pid***, IX_MBUF *****pMblk*** , BOOL ****discard***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| pid | The port where the frame is being sent. | I |
| **pMblk | The mbuf for that frame. IX_MBUF for VxWorks, sk_buff for Linux. | I |
| *discard | TRUE if VLAN module makes decision to drop this frame. | O |
| **Return:** | IX_FAIL – Could not add VLAN tag info, not member of Egress VLAN, Illegally tagged frame.<br>IX_SUCCESS – Transmit the frame. | |
| **Description:** | This function is used to check a frame against applicable filters, VLAN membership, apply VLAN tag information and submit to network driver. | |

## 6.1.8 VLAN Module Types

| TypeDef | Type | Description |
|---|---|---|
| ACCEPT_TYPE | enum | ALL_FRAME, ONLY_TAGGED_FRAME |
| DIRECTION | enum | INGRESS, EGRESS |
| GROUP_ID | unsigned short | 0-15 |
| IP_ADDRESS[4] | unsigned char | |
| MAC_ADDRESS[6] | unsigned char | |
| PORT_BITMAP | unsigned short | Bit 0 for port 0, and so on |
| PORT_ID | unsigned short | 0,1 |
| PRIORITY | unsigned short | 802.1q priority level. Acceptable values: 0 to 7. |
| RULE_ID | unsigned short | |
| VLAN_ID | unsigned short | Acceptable values: 0 to 4095 |
| PROTOCOL_TYPE | signed long | TCP=0x06, UDP=0x11. |
| PORT_NUM | signed long | TCP or UDP protocol port number. |

| Struct: | IP_RULE | |
|---|---|---|
| **TypeDef:** | **Elements:** | **Description:** |
| VLAN_ID | vid | |
| PRIORITY | priority | |
| IP_ADDRESS | src_ip, src_ip_mask | |
| IP_ADDRESS | dst_ip, dst_ip_mask | |
| PROTOCOL_TYPE | protocol | |
| PORT_NUM | src_port | |
| PORT_NUM | dst_port | |

| Struct: | MAC_RULE | |
|---|---|---|
| **TypeDef:** | **Elements:** | **Description:** |
| VLAN_ID | vid | |
| PRIORITY | priority | |
| MAC_ADDRESS | src_ip | |
| MAC_ADDRESS | src_ip_mask | |

## 6.2 Ingress QoS Module

### 6.2.1 General Control Path Interface

| **Prototype:** | IX_STATUS **ixIngressQosInit** ( UINT8 ***numTrafficClass*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| numTraffic-Class | number of traffic classes (number of priority queues) to use for Ingress Qos. The range of numTrafficClass is from 1 through 8, inclusively. | I |
| **Return:** | IX_FAIL<br>IX_SUCCESS | |
| **Description:** | This function is used to initialize the Ingress QoS module with a number of traffic classes. This in turn sets up numTrafficClass ingress queues, the priority mapping, and initializes default shaper utilities. The threshold of each ingress priority queue, and the priority mapping which maps (port, vlan_priority) to (traffic class) are set to default states.<br>The default priority mapping consists of one real-time traffic class (real-time_traffic_class = numTrafficClass-1. Higher numeric value means higher priority), equivalent number of traffic class 0, 1, 2, …, (real-time_traffic_class -1), with exception that traffic class 0 may have more mappings. For example, if numTrafficClass=4, then real-time_trafffic_class=3, and the mapping for port 0, vlan_priority (0,¡K,7) to traffic class is like ((port, vlan_priority)->(tc)): (0,0) -> (0), (0,1) -> (0), (0,2) -> (0), (0,3) -> (1), (0,4) -> (1), (0,5) -> (2), (0,6) -> (2), (0,7) -> (3).<br>For each ingress priority queue, there are low threshold and high threshold associated with each queue. This function initializes these queue and their corresponding thresholds with default values. Those defaults are configurable in header file. Currently only high threshold are of significant to IxIngress Qos, low threshold is left for future implementation. If the number of frames buffered in the ingress priority queue reaches high threshold, that frames are dropped. The number of traffic classes and the number of ingress queues are equivalent in IxIngress QoS module and are used interchangeably.<br>This function only initializes basic shaper utilities, it does not configure any shaper and does not enable the timer. The default settings of shapers and timers are to use maximum allowed parameters for shapers, and to use 100ms timer task. Since there are no shapers being configured in this function, frames are still processed by original Ethernet driver after invoking this function. | |

| Prototype: | IX_STATUS **ixIngressQosDown** ( void ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| n/a | | |
| **Return:** | IX_FAIL<br>IX_SUCCESS | |
| **Description:** | Bring down the Ingress QoS module.<br>All active shapers are initialized to their initial states and then disabled. Timer, whether in form of task or interrupt, is disabled. All frames buffered in ingress priority queues are flushed out. The priority mapping is also initialized to its default state. Frames received after the invocation of this function are processed by original Ethernet driver. | |

## 6.2.2 802.1p to Traffic Class Interface

| Prototype: | IX_STATUS **ixQos1pPriority2TCMapSet** ( UINT8 ***port***, UINT8 ***priority***, UINT8 ***tc*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| port | Port number 0 or 1. | I |
| priority | Vlan Priority, 0-7. | I |
| tc | Traffic Class. Must be between 0 and [***numTrafficClass***-1] | I |
| **Return:** | IX_FAIL<br>IX_SUCCESS | |
| **Description:** | Set the traffic class according to (port, vlan_priority).<br>This function sets up priority mapping of (port, vlan_priority) -> (traffic class). This mapping is first setup when IxIngressQosInit(numTrafficClass) is invoked. The mapping set here overwrites the previous setting. The mapping can be set on the fly when the traffic is still flowing. The input parameter should be in legal range; notably the tc parameter must not exceed numTrafficClass set previously. There is no constraint as to how the mapping is defined; multiple (port, vlan_priority) can map to the same (tc). This function returns IX_FAIL if IxIngress QoS module is not enabled by (IxIngressQosInit). | |

| Prototype: | IX_STATUS **ixQos1pPriority2TCMapGet** ( UINT8 ***port***, UINT8 ***priority***, UINT8 ****tc*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| port | Port number 0 or 1. | I |
| priority | Vlan Priority, 0-7. | I |
| *tc | Traffic Class. Cannot be NULL. | O |
| **Return:** | IX_FAIL<br>IX_SUCCESS | |
| **Description:** | Get the traffic class according to (port, vlan_priority).<br>This function obtain the traffic based on the inputs of (port, vlan_priority) pair. The *tc cannot be NULL, otherwise the function returned IX_FAIL. This function is mostly used for inquiring traffic class for some (port, vlan_priority) combination. This function returns IX_FAIL if IxIngress QoS module is not enabled by (IxIngressQosInit). | |

| Prototype: | IX_STATUS **ixQos1pPriority2TCMapShow** ( UINT8 ***buf*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *buf | Pointer to buffer receiving the map | O |
| **Return:** | IX_FAIL<br>IX_SUCCESS | |
| **Description:** | Show the traffic class mapping.<br>The shown message can be referred from *buf pointer. The caller can use *buf to obtain the messages exported from IxIngress QoS module. The message shown includes (port, vlan_priority)->(tc) mapping, as well as the low and high threshold for each ingress queue. If the compilation option IX_QOS_STAT is defined, it also shows total number of frames classified to different traffic classes, the number of frames currently buffered in the ingress priority queue, the total number of frames that are been put in a particular ingress queue and then transmitted, and the total number of frames that are dropped.<br>The priority mapping, high/low threshold, and the statistics are reset when the QoS module is disabled ( IxIngressQosDown() ) and then enabled ( IxIngressQosInit() ). | |

## 6.2.3 Ingress Queue Interface

| Prototype: | IX_STATUS **ixQosIngressQIsEmpty** ( UINT8 ***q*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| q | Priority Queue Number. It should be between 0 and real_time_traffic_class. | I |
| **Return:** | IX_FAIL – Queue contains at least one frame<br>IX_SUCCESS – Queue is empty | |
| **Description:** | Check if a particular priority queue is empty.<br>This function is used both in data path and in control path, thus is does not perform a check for input parameter q. The caller from control path should be aware of this and check the parameter before invocation. | |

| Prototype: | IX_STATUS **ixQosIngressQNumQGet** ( UINT8 ***numQ** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *numQ | Number of priority queues. | O |
| **Return:** | IX_FAIL – IxIngress QoS module not intialized, no queues exist.<br>IX_SUCCESS | |
| **Description:** | Get the number of priority queues.<br>This function obtains the number of ingress priority queues in IxIngress QoS module. It must be called only after the IxIngress QoS module is initialized; otherwise IX_FAIL is returned. | |

| Prototype: | IX_STATUS **ixQosIngressQThresholdSet** ( UINT8 ***q***, UINT16 ***low***, UINT16 ***high*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| q | Priority Queue Number. It should be between 0 and real_time_traffic_class. | I |
| low | Low Threshold. Must be > 0 and < total number of available mbufs allowed. | I |
| high | High Threshold. Must be >= 0 and < total number of available mbufs allowed. | I |
| **Return:** | IX_FAIL – Ingress QoS Priority queue not initialized, q out of range, threshold out of range.<br>IX_SUCCESS | |
| **Description:** | Set the watermark threshold for priority queue q.<br>This function sets the low and high threshold for a particular ingress priority queue q. In the current design only high threshold is used. If the number of frames buffered in a particular ingress priority queue reaches high threshold, no frames are allowed to be buffered in this queue until some frames are transmitted. The rationale behind this design is to ease the headroom congestion of mbuf in the memory pool, so that NPEs can still get some mbufs when there are higher priority or real-time traffic influx from networks. The high/low threshold has no significance for some ingress priority queues, including those for real-time traffic class, and those classes whose shapers are not configured (those queues are called inactive ingress queues, in contrast to active ingress queues where frames may be buffered in those queue if tokens in corresponding shapers are depleted). Note if the total of high thresholds of all active queues are more than total available mbuf allowed in the system, higher priority or real-time traffic may be rejected by the NPE. One scenario that makes this happen is when the input rates for all active queues (traffic classes) are much higher than their corresponding shapers' rate. All available mbuf are buffered in ingress priority queues and the NPE cannot obtain any free mbuf from the free mbuf pool. In other words, all mbuf's are stuck in IxIngress QoS ingress priority queue, and higher priority or real-time traffic are dropped partially or entirely, depending on the traffic condition.<br>It is therefore the upper layer's responsibilityt to make arrangements for those high/low thresholds to meet the expected QoS requirement. Larger high thresholds allow more tolerance for traffic perturbation when the input rate are about the configured rate, but provide less room for other queues. Alternatively, smaller high thresholds pose more strict conditions for input traffic, but allow more flexibility for other queues. Applications with real-time or interactive properties may map to higher priority traffic with smaller high threshold; applications that are of best-effort properties are better mapped to lower priority traffic with a larger high threshold. | |

| Prototype: | IX_STATUS **ixQosIngressQThresholdGet** ( UINT8 ***q***, UINT16 ****low***, UINT16 ****high*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| q | Priority Queue Number. It should be between 0 and real_time_traffic_class. | I |
| *low | Low Threshold. Cannot be NULL. | O |
| *high | High Threshold. Cannot be NULL. | O |
| **Return:** | IX_FAIL – Ingress QoS Priority queue not initialized, q out of range.<br>IX_SUCCESS | |
| **Description:** | Get the watermark threshold for priority queue q.<br>This function is used to obtain the low and high thresholds for a particular ingress priority queue. It does not matter if the corresponding traffic class (ingress queue) is real-time traffic class, or the shaper for that traffic class is not configured. | |

## 6.2.4 Ingress Traffic Shaper Interface

| Prototype: | IX_STATUS **ixQosIngressShaperInit** ( void ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| n/a | | |
| **Return:** | IX_FAIL – Shaper module is already initialized, Ingress QoS Priority mapping or Priority queue not initialized, timer failed to initialize.<br>IX_SUCCESS | |
| **Description:** | Initialize the shaper module. The Ingress QoS mapping and priority queues must be initialized before the shaper module. This function will also initialize the timer, but the timer will not be enabled (and traffic wil not use the shaper) until at least one shaper has been configured. | |

| Prototype: | IX_STATUS **ixQosIngressShaperDown** ( void ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| n/a | | |
| **Return:** | IX_FAIL – Shaper module is not initialized.<br>IX_SUCCESS | |
| **Description:** | Bring down the shaper module. All active shapers are reset. Traffic flows as if there were no shaper. | |

| Prototype: | IX_STATUS **ixQosIngressShaperCfgSet** ( UINT8 ***q***, UINT32 ***avgD***, UINT32 ***avgF***, UINT32 ***PeakD***, UINT32 ***PeakF***, UINT32 ***type***, UINT32 ***parMA***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| q | Ingress shaper queue. It should be between 0 and real_time_traffic_class. | I |
| avgD | Average data bytes per second. Range [0 – (**100*1,000,000**/8)] The notation described is the maximum number of uni-directional bytes per second on a 100Mbps interface. | I |
| avgF | Average frame count per second. Range [0 – (**100*1,000,000**/8/64)]. The notation described is the maximum number of uni-directional 64-byte Ethernet frames per second on a 100Mbps interface. | I |
| peakD | Peak data bytes per second. Range [avgD – (**100*1,000,000**/8)]. The notation described is the maximum number of uni-directional bytes per second on a 100Mbps interface. | I |
| peakF | Peak frame count per second. Range [avgF – (**100*1,000,000**/8/64)] The notation described is the maximum number of uni-directional 64-byte Ethernet frames per second on a 100Mbps interface. | I |
| type | Type of shaper. 1 – data byte, 2 – frame count, 3 both. | I |
| parMA | Parameter for moving average. Not implemented. | I |
| **Return:** | IX_FAIL – Ingress QoS shaper not initialized, inputs out of range.<br><br>IX_SUCCESS | |
| **Description:** | Configure the shaper for a particular traffic class (ingress queue).<br>This function is used to configure the shaper for a particular traffic class. In the current design, every traffic class has a corresponding shaper. By default, when the IxIngress QoS module is first initialized, the states for all shapers are set to default and are disabled (not configured). Invoking this function enables and configures one of the shapers. If this is the first shaper being configured, then traffic entering from the network goes into the IxIngress QoS module after the invocation of this function. In the meantime, timers are enabled and running after the first shaper is configured. If there are already some shapers running, this function only configures a new shaper (if the shaper for that traffic is not configured yet), or modifies the old setting of originally configured shaper (if the shaper for that traffic class already configured). The configuration or re-configuration of shapers can be performed on the fly, and it is not required to stop traffic or to disable IxIngress QoS module before configuration or re-configuration. The new configuration takes effect immediately after the next timer slot (refer to the timer module).<br>The design of the shaper follows the notation of token bucket, where each shaper of token bucket comprises a bucket of depth *depth*, and a steady flow of tokens that influx into the bucket at rate *rate*. Frames are allowed to transmit whenever there are available tokens in the bucket. The transmitted frame also takes out tokens. The number of tokens consumed by a frame is determined by the type of shaper. Currently there are two types of shapers implemented: Data-byte type, and Frame-count type. For data-byte shaper, a frame takes *length-of-frame* tokens, while for frame-count shaper, one frame takes one token. Shapers are updated at the beginning of timer slot, and the duration of the time slot is determined by timer APIs described in this document. By default, it is 100 ms. Updating the shaper basically increases the number of tokens by *rate*. If the number of tokens in the bucket after the update is more than *depth*, then the total number of tokens in the bucket is limited to *depth*. If, on the contrary, the number of tokens after the update is still below zero (which might be the case when one large frame takes lots of tokens, given that the number of available tokens are just above zero), then no frames are allowed to pass the shaper in the next timer slot.<br>Note that if an upper-layer application uses only one type of shaper, then only the corresponding set of parameters are used, and the other set of parameters are ignored. The moving average of queue status is not currently implemented. | |

| **Prototype:** | IX_STATUS **ixQosIngressShaperCfgGet** ( UINT8 ***q***, UINT32 ****avgD***, UINT32 ****avgF***, UINT32 ****PeakD***, UINT32 ****PeakF***, UINT32 ****type***, UINT32 ****parMA*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| q | Ingress shaper queue. It should be between 0 and real_time_traffic_class. | I |
| *avgD | Average data bytes per second returned from IxIngress QoS module. Cannot be NULL. | O |
| *avgF | Average frame count per second returned from IxIngress QoS module. Cannot be NULL. | O |
| *peakD | Peak data bytes per second returned from IxIngress QoS module. Cannot be NULL. | O |
| *peakF | Peak frame count per second returned from IxIngress QoS module. Cannot be NULL. | O |
| *type | Type of shaper. 1 – data byte, 2 – frame count, 3 both returned from IxIngress QoS module. Cannot be NULL. | O |
| *parMA | Parameter for moving average. Not implemented. Cannot be NULL. | O |
| **Return:** | IX_FAIL – Ingress QoS shaper not initialized.<br>IX_SUCCESS | |
| **Description:** | Get the shaper configuration for a traffic class.<br>This function is used for obtaining the shaper configuration. | |

| **Prototype:** | IX_STATUS **ixQosIngressShaperTypeChange** ( UINT8 ***q***, UINT32 ***typeNew*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| q | Ingress shaper queue. It should be between 0 and real_time_traffic_class. | I |
| typeNew | New type of shaper to be used. 1 – data byte, 2 – frame count, 3 both. | I |
| **Return:** | IX_FAIL.<br>IX_SUCCESS | |
| **Description:** | Configure the shaper type of a pre-existing shaper.<br>This function is used to modify the type of shaper for a traffic class. It only changes the type of the shaper but keeps all other parameters intact. For example, the upper layer application may first configure the shaper by calling ***IxQosIngressShaperCfgSet***(…) with parameters of aBPS/aFPS, pBPS/pFPS, and designated type of Data-Byte shaper. The aFPS/pFPS pair are not used if the shaper type is Data-Byte. Later on it may change the type to Frame-count shaper. And in this case, aFPS/pFPS are used and aBPS/pBPS are ignored. This function can be invoked in the run time. The new type of shaper takes effect immediatly after the next time slot. All parameters must be in their legal range, otherwise IX_FAIL is returned. If the ***newType*** is identical to the original setting, nothing will happen. | |

| Prototype: | IX_STATUS **ixQosIngressShaperCfgReset** ( UINT8 ***q*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| q | Ingress shaper queue. It should be between 0 and real_time_traffic_class. | I |
| **Return:** | IX_FAIL – Ingress QoS Shaper not initialized, Shaper q is not configured, q is out of range.<br>IX_SUCCESS | |
| **Description:** | Reset the shaper configuration for a traffic class shaper.<br>This function resets the shaper for a traffic class, if the shaper for the traffic class is previously configured. After the reset, the shaper parameters are reset to default, and that shaper is disabled. If after resetting the shaper, there is no active shaper in the system, then the timer is disabled, and all traffic will be handled by Ethernet driver as if there were no IxIngress QoS module.<br>Upon the reset of the shapers, frames buffered in the corresponding queue are dropped.<br>Traffic does not need to be stopped before issuing this function. There is no effect if the shaper to be reset is not enabled. | |

| Prototype: | IX_STATUS **ixQosIngressShaperCfgShow** ( UINT8 ****buf*** ); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *buf | Buffer to contain shaper configurations and statistics. Cannot be NULL. | O |
| **Return:** | IX_FAIL – Ingress QoS Shaper not initialized.<br>IX_SUCCESS | |
| **Description:** | Show the shaper configurations of all active shapers.<br>The shown message can be referred from ****buf*** pointer. The caller can use ****buf*** to obtain the messages exported from IxIngress QoS module. The message shown (per active shaper) includes type of the active shaper, aBPS, aFPS, pBPS, pFPS, quota available in the current time slot, number of transmitted bytes/ frames in the current time slot. The timer resolution and timer type etc. | |

## 6.2.5 Timer Configuration Interface

| Prototype: | IX_STATUS **ixQosIngressTimerCfgSet** ( UINT8 ***type***, UINT16 ***msNew***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| type | Type of timer. 1 – timer task, 2 – watchdog timer interrupt. | I |
| msNew | New timer resolution in ms. To change timer type but keep original resolution, use 0. | I |
| **Return:** | IX_FAIL<br>IX_SUCCESS | |
| **Description:** | Configure new type of timer or new timer resolution to use.<br>This function is used to set new timer type (when ***msNew*** = 0), or set new timer type and new timer resolution. There are two types of timer available for use: timer task and watchdog timer interrupt. The benefit of using timer task is it does not consume extra system resource to give one timer, but the resolution is limited to 1 OS tick (in VxWorks, it is 17 ms, in Linux, it is 10 ms), and accuracy suffers when the system is busy. Even though it consumes system resources, the watchdog interrupt timer is more accurate and there is no limit on the timer resolution. By default, the timer type is timer task with a coarse resolution of 100 ms.<br>This function must be invoked after the initialization of IxIngress QoS module. It is, however, not required to be called after some shapers are configured. If this function is called when some shapers are active, in which case the timer is running, the new setting first disables the old timer, re-configures the new timer, and then enables the new timer, resulting in the timer running. If there are no shapers being configured at that instant (that is, the timer is not enabled), then calling this function only changes the parameters to be used. The timer is enabled automatically when the first shaper is configured. | |

| Prototype: | IX_STATUS **ixQosIngressTimerCfgGet** ( UINT8 ***type***, UINT16 ***ms***, UINT32 ***id***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *type | Type of timer. 1 – timer task, 2 – watchdog timer interrupt. Cannot be NULL. | O |
| *ms | Timer resolution in ms. Cannot be NULL. | O |
| *id | Timer task ID. Does not apply to watchdog timer interrupt type. | O |
| **Return:** | IX_FAIL<br>IX_SUCCESS | |
| **Description:** | Get new configuration of the timer<br>This function is used to get the configuration of timer parameter. It does not matter whether the timer is running or not. It must be called only after IxIngress QOs module is initialized. | |

| Prototype: | IX_STATUS **ixQosIngressTimerCfgRemove** ( void ); | |
|---|---|---|
| **Parameters:**<br>n/a | **Description:** | **I/O:** |
| **Return:** | IX_FAIL<br>IX_SUCCESS | |
| **Description:** | Remove the timer<br>This function is used to remove the timer. As a consequence, all the active shapers are disabled. While explicitly calling this function forcefully dismisses all running shapers, this function may be invoked implicitly when all the active shapers are disabled. If this function is called before IxIngress QoS module is initialized, it returns IX_FAIL. There are no effect when this function is called and no shapers are configured. | |

## 6.2.6 Ingress QoS Data Path

| Prototype: | IX_STATUS **ixQosIngressProcess** ( IX_MBUF ***pMblk***, UINT8 ***port***, UINT8 ***priority***, UINT32 ***drop_cnt***, UINT32 ****pkt_cnt***, UINT32 ****byte_cnt***, UINT32 ***len***); | |
|---|---|---|
| **Parameters:** | **Description:** | **I/O:** |
| *pMblk | The mbuf for that frame. IX_MBUF for VxWorks, sk_buff for Linux. | IO |
| port | The port where the frame enters. | I |
| priority | The VLAN priority obtained from VLAN module. If VLAN is not enabled, this parameter is 0. | I |
| drop_cnt | The number of frames being dropped by the driver. NULL for VxWorks. | I |
| *pkt_cnt | The number of frames received by the driver. NULL for VxWorks. | O |
| *byte_cnt | The number of bytes transmitted by the driver. NULL for VxWorks. | O |
| len | The length of the frame. NULL for VxWorks. | I |
| **Return:** | IX_SUCCESS – the frame was successfully sent to the IxIngress QoS module.<br>IX_FAIL – the frame failed to be sent to the IxIngress QoS module. | |
| **Description:** | This function is the entry point of IxIngress QoS module. The current design works for Linux and VxWorks. It does not perform any input parameter checking. The caller is accountable for the validity of the input parameters. All the parameters are used in Linux. In VxWorks, only the first three are of significance. The last four parameters can be NULL. | |

| | | |
|---|---|---|
| **Prototype:** | IX_STATUS **ixIngressQosShaperConfigDone** ( void ); | |
| **Parameters:**<br>n/a | **Description:** | **I/O:** |
| **Return:** | IX_FAIL – no shaper has been configured.<br><br>IX_SUCCESS | |
| **Description:** | Check if there is at least one shaper configured. If this is the case, then all frames that hit the Ethernet driver go into IxIngress QoS module. If there is no shaper configured, frames are processed by Ethernet driver's original data path. | |