RTL839x
LAYER 2 PLUS MANAGED 52*10/100/1000M-PORT SWITCH CONTROLLER

RTL835x
LAYER 2 PLUS MANAGED 52*10/100-PORT SWITCH CONTROLLER

Developer Guide

Rev. 1.5
3 December 2013

Realtek Semiconductor Corp.
No. 2, Innovation Road II, Hsinchu Science Park, Hsinchu 300, Taiwan
Tel.: +886-3-578-0211 Fax: +886-3-577-6047
www.realtek.com

USING THIS DOCUMENT
This document is intended for use by the system engineer when integrating with Realtek switch products. Though every effort has been made to assure that this document is current and accurate, more information may have become available subsequent to the production of this guide. In that event, please contact your Realtek representative for additional information that may help in the development process.

Revision	Release Date	Summary
1.0	2012/11/15	Initial Release
1.1	2012/11/29	Refine "L2 Switching" and "Traffic Isolation" chapters
1.2	2013/01/02	1.Refine all chapters 2.Update ACL chapter - Modify Pre-Defined Template 0, 1, 4 - Modify definition of template field type SPM3 and IP_LEN_RANGE
1.3	2013/01/21	1.Add "VLAN Translation Application Example" section 2.Update OAM Dying Gasp section
1.4	2013/03/11	Update the description of section 4.4.3 VLAN Based Learning Constraint
1.5	2013/12/03	Update the description of section Cable Diagnostic

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

The RTL8390 chip family is a multi-layer Fast/Gigabit Ethernet switch devices which integrates a MIPS34Kc CPU subsystem running in 700MHz for 839x and 650MHz for 835x. The RTL8390 is a cost-effective solution for Small-Medium Business (SMB) and Carrier Ethernet Access/Edge applications with wire-speed performance for 24/48 FE, 24/48 GE and 24GE+2*10GE platforms.

1.1 Switching Family Overview

The RTL8390 is the REALTEK third generation Layer 2 Plus Managed Switch which contains the family members listed below:

Table 1: RTL8390 Family Member

Chip Name	FE Ports	GE Ports	10GE Ports
RTL8353	48	4	X
RTL8392	X	28	X
RTL8393	X	52	X
RTL8396	X	24	2

1.2 Family Feature Overview

The RTL8390 major features including CPU subsystem and their capacity are shown in below table.

Table 2: Feature Summary

Feature	Desctiption
CPU sub-system	
CPU	Integrated 32bit MIPS-34Kc 700Mhz for 839x 650Mhz for 835x
L1 I/D cache	32KB/32KB
on-chip SRAM	96KB
Flash Interface	SPI(Single/Dual/Quad) (up to 128MB)
Memory Interface	DDR I/II/III (up to 256MB)
UART	2*UART (UART2 shared with EJTAG)
Hardware Capability and Interface	
Integrated PHY Chipset	X
Switch Capacity	24G+4G or 48G+4G or 48FE+4G
Packet Buffer	12Mbit
Jumbo Frame	12KB
Management port Inteface	SGMII for external CPU
MAC and Port Capability	
MAC Address Table Size	16K(4-way hash)
Multicast Table Size	4K
Aging and New learnt notification	Offload SW effort for maintaining a synchronous L2

	table with HW
Mirror & Sampling	
Number of mirroring set	4
RSPAN	Sourece, Intermediate, Destination
Sampling(sflow)	Ingress Port, Egress Port
IPFIX	V
VLAN	
1Q and QinQ VLAN (S tag aware)	4K
Protocol VLAN	8
Mac-based VLAN	V(share with 1K Ingress VLAN Translation)
IP-Subnet-based VLAN	V(share with 1K Ingress VLAN Translation)
VLAN translation	1K Ingress, 1K Egress
N:1 VLAN translation	V (via MAC Address Table)
VLAN Profile	8
Protocols	
Trunk (IEEE802.3ad LACP)	
Groups	16
Hash Method Configurable for Load Balance	SPN/SMAC/DMAC/SIP/DIP/SPORT/DPORT
Ports for Each Group	8
Trunking Fail over	V
Multicast	
IGMP/MLD ASM snooping	V
IGMP/MLD SSM snooping	V
Lookup Method	(DMAC,VID), (GIP, VID), (GIP, SIP), (GIP, SIP, VID)
IPv6 multicast lookup using selective SIP/DIP byte	V
Number of Groups	Up to 16K Groups but only 4K different Port Mask
Bridge	
Spanning tree algorithm	STP/RSTP/MSTP
Multiple spanning tree instance	256
H/W Loop Detection/Prevention	X
QoS	
Amount of Queues	Per port 8 egress queues
Storm Control	V(bps or pps)
Control Protocol Strom Control	V (BPDU, ARP, IGMP)
Ingress Bandwidth Control	V
Egress Bandwidth Control	V(rate control for CPU port)
Scheduling method	WRR/WFQ/Strict/Strict+WFQ/Mixed
WRED	Simple WRED
ACL	
Number of Entries	2K+256
Entry Width	216-bits
Template Number	total 8 template(5 pre-defined, 3 user defined)
Ingress ACL	V

Egress ACL	V
Meter/Marking	512(DLB/srTCM/trTCM)
Counter	2K(packet-based)/1K(byte-based)
Multi-hit Log counter	V (different ACL block)
Range Check	
VLAN	32
IP Address	8 IPv4 or 2 IPv6 or 4 IPv6 64b
L4 PORT	8 (shared by VLAN/L4 Port/Packet Length)
Packet Length	8
Key	
ARP/RARP header	V
IPv6 Address	V
Source Port Mask	V
Field Selector	Global 12 groups(16-bit per group)
Security	
H/W-based 802.1X	X
MAC learning constriant	System/Port/VLAN
DoS prevention	L2~L4
MAC White List	V
Layer 3/Tunneling	
Static IPv4/IPv6 routing	V(ACL)
IP over MPLS	LER/LSR, (ACL, 256 Dobule Label/512 single Label)
Management/Carrier Ethernet	
H/W OAM Loopback	V
H/W OAM Dying Gasp	Software-Defined Payload
802.1ag CFM	V(G.8031/8032)
Y.1731 CFM	ETH-DM
Power Saving	
EEE	V(supported by PHY)
AVB(Audio/Video Bridging)	
H/W PTP timestamp	V(supported by PHY)

2 Architecture Overview

2.1 Device Block Diagram

The RTL8390 family supports 14 SERDES in total to connect to Fast/Gigabit Ethernet PHYs. R-XAUI interface is supported for 10G PHY connection while 10G-R (XFP/SFP+) interface is supported for direct 10G fiber connection without connecting a 10G PHY between. The maximum wire-speed bandwidth for RTL8390 is 52Gbps.

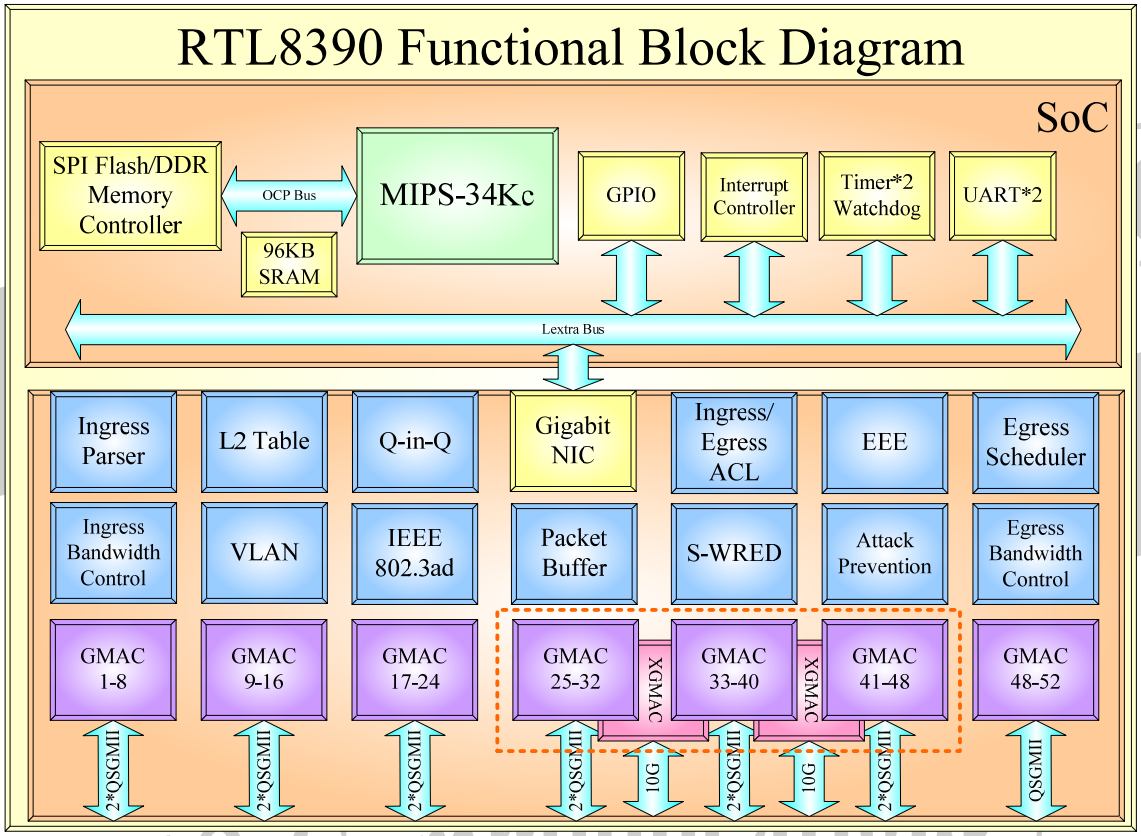


Figure 1: RTL8390 Block Diagram

2.2 Pipeline WalkThrough

The packet goes through the modules in below sequence.

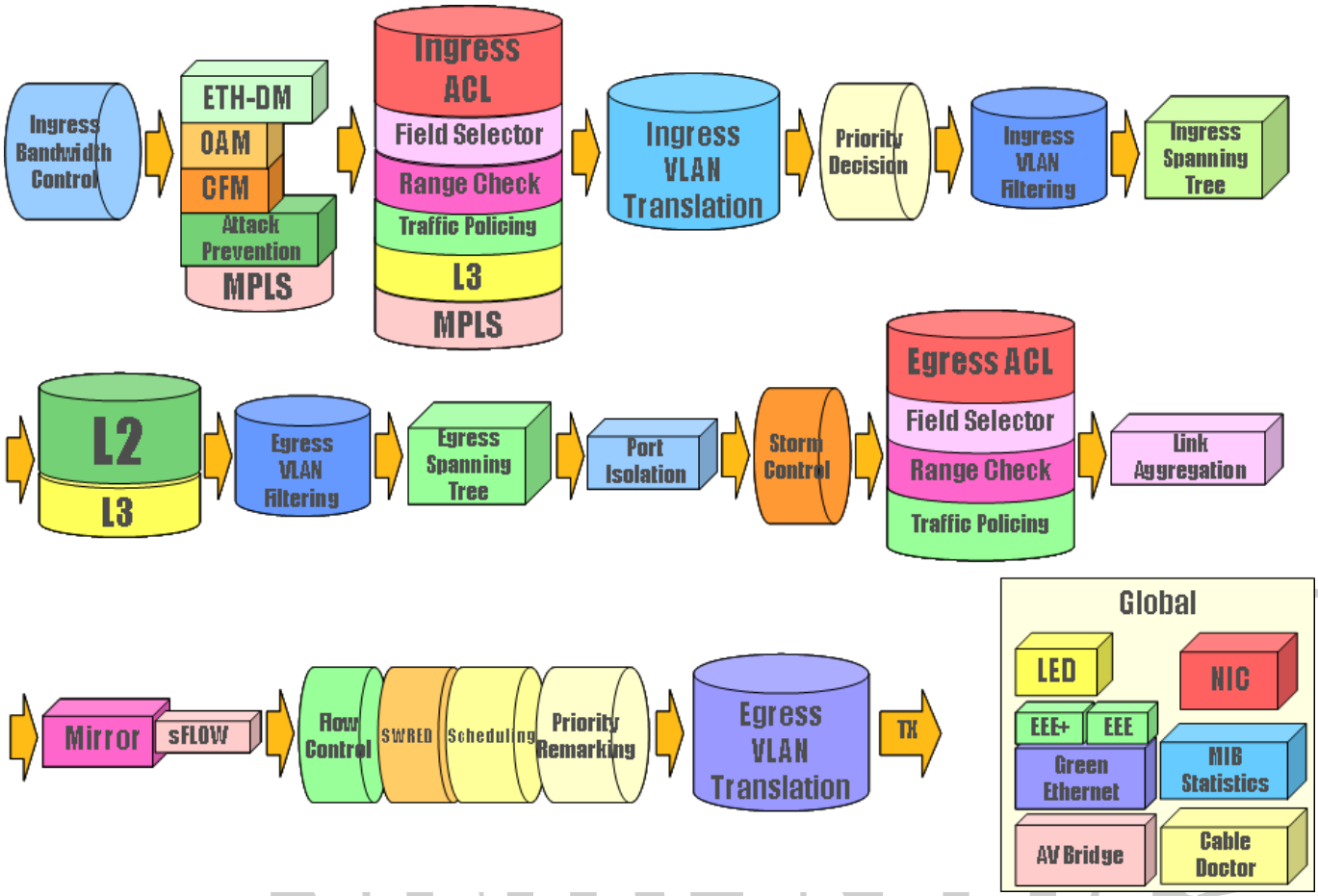


Figure 2: Packet Processor Pipeline

3 Virtual Local Area Network

VLANs are used to segment the network into smaller broadcast domain or segments. The primary reason to segment the network is to relieve network congestion and increase bandwidth. The device supports up to 4096 (0-4095) VLANs which are used by inner and outer VLAN. Each VLAN can specify the member ports, untag ports for VLAN filtering and VLAN Tag manipulation. It supports flexible VLAN tag manipulation for different VLAN translation and Q-in-Q applications, such as, 1:1 VLAN translation, N:1 VLAN translation and MAC-based N:1 VLAN translation. In addition to support port-based VLAN and 1Q/1ad VLAN, the device also supports protocol-based, IP subnet-based, MAC-based and flow-based VLAN.

3.1 VLAN Functional Blocks

Packet goes over the VLAN functional blocks in below sequence.

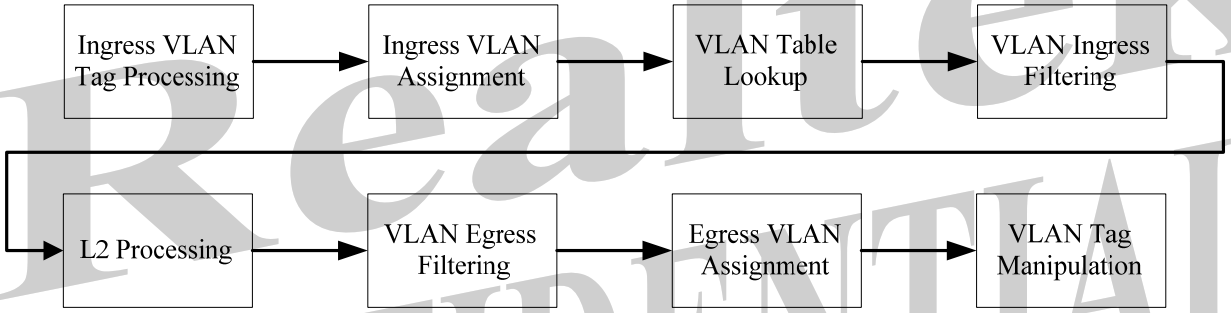


Figure 3: VLAN Functional Blocks Flow

The device first parses the VLAN tag of the packet. The packet is then examined by Accept Frame Type configurations (VLAN_PORT_ACCEPT_FRAME_TYPE register). Ingress VLAN assignment module determines the forwarding VLAN ID which is used for VLAN table lookup. After obtaining the VLAN member ports, the packet is examined by VLAN ingress and egress filtering modules. Before the packet is transmitted, egress VLAN assignment module determines the VLAN ID that is going to be encapsulated in VLAN tag. The egress VLAN ID can be the VLAN ID the device received or a new one assigned by other modules, such as ACL. VLAN tag manipulation module finally determines the VLAN tag status of the packet.

3.2 Ingress VLAN Tag Processing

The device supports up to three layer VLAN tags, namely outer (S-Tag), inner (C-Tag) and extra tag to cover different VLAN applications. In traditional VLAN application, only one VLAN tag is used and usually configures inner tag to be C-Tag (0x8100). In Q-in-Q application, two VLAN tags are used and usually configure inner tag to be C-Tag (0x8100) and outer tag to be S-Tag (0x88A8). Three VLAN tags are used for supporting the Q-in-Q application which may receive S+C tagged packets from downlink port and need to insert a third tag to the packet before transmitting it to the uplink port. In this case, the device should configure the inner tag as S-Tag and extra tag as C-Tag and the device can then insert the outer tag (third tag) to the packet according to the ingress port or the S-VID.

The TPID of three VLAN tags are configurable. The device supports four global outer TPIDs configured by VLAN_TAG_TPID_CTRL.OTPID fields, four global inner TPIDs configured by VLAN_TAG_TPID_CTRL.ITPID fields and one global extra TPID configured by VLAN_ETAG_TPID_CTRL.ETPID filed. Each ingress port can specify which TPIDs are used for parsing the packets through VLAN_PORT_OTAG_TPID_CMP_MSK, VLAN_PORT_ITAG_TPID_CMP_MSK, VLAN_PORT_ETAG_TPID_CMP registers. The VLAN tag processing flow is illustrated below:

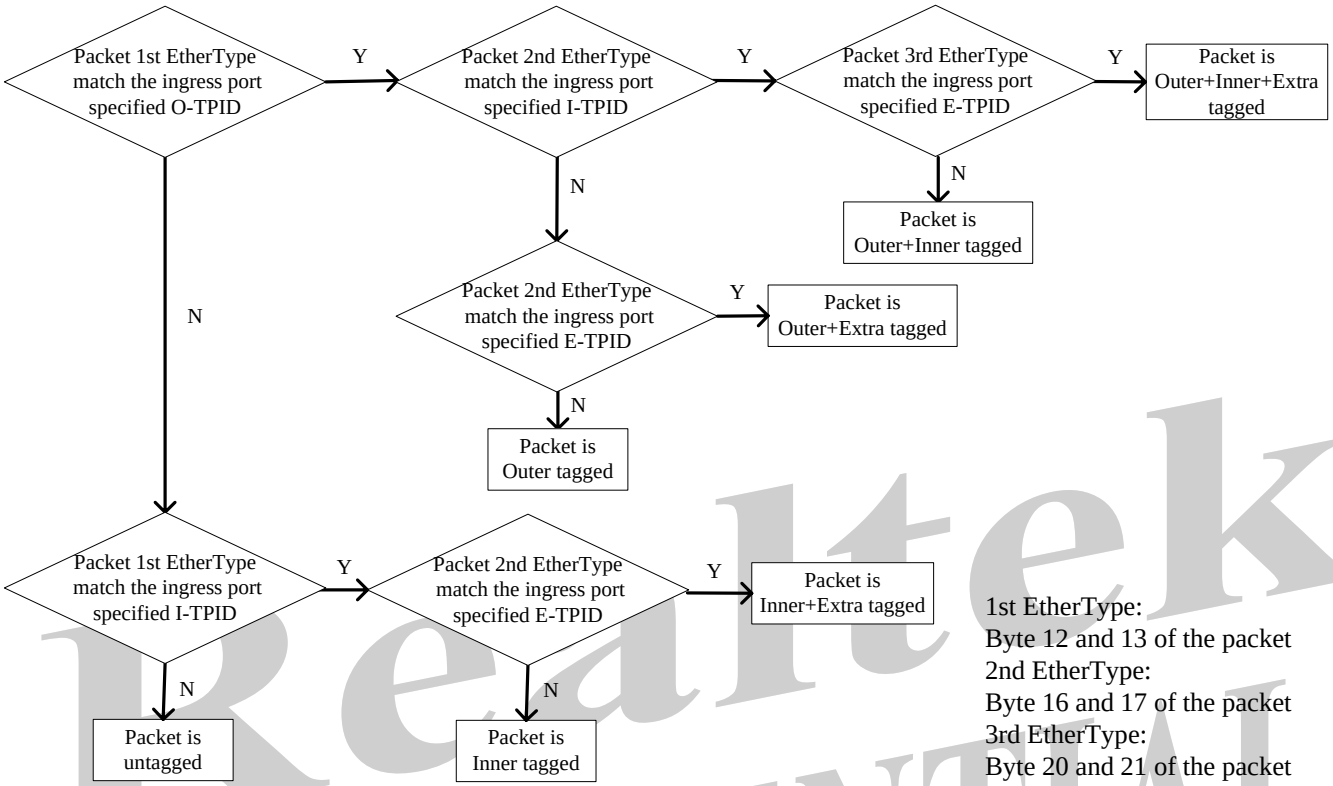


Figure 4: Ingress VLAN Tag Processing Flow

Extra tag is not used inside the device but recognized by the parser. Extra tag can only be parsed for the following tag combinations: Outer+Inner+Extra, Outer+Extra, Inner+Extra.

API REFERENCE

```
rtk_vlan_innerTpidEntry_get(uint32 unit, uint32 tpid_idx, uint32 *pTpid);
rtk_vlan_innerTpidEntry_set(uint32 unit, uint32 tpid_idx, uint32 tpid);
rtk_vlan_outerTpidEntry_get(uint32 unit, uint32 tpid_idx, uint32 *pTpid);
rtk_vlan_outerTpidEntry_set(uint32 unit, uint32 tpid_idx, uint32 tpid);
rtk_vlan_extraTpidEntry_get(uint32 unit, uint32 tpid_idx, uint32 *pTpid);
rtk_vlan_extraTpidEntry_set(uint32 unit, uint32 tpid_idx, uint32 tpid);

rtk_vlan_portIgrInnerTpid_get(uint32 unit, rtk_port_t port, uint32 *pTpid_idx_mask);
rtk_vlan_portIgrInnerTpid_set(uint32 unit, rtk_port_t port, uint32 tpid_idx_mask);
rtk_vlan_portIgrOuterTpid_get(uint32 unit, rtk_port_t port, uint32 *pTpid_idx_mask);
rtk_vlan_portIgrOuterTpid_set(uint32 unit, rtk_port_t port, uint32 tpid_idx_mask);
rtk_vlan_portIgrExtraTagEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);
rtk_vlan_portIgrExtraTagEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
```

3.3 Ingress VLAN Assignment

The device assigns two ingress VLAN IDs (inner and outer VLAN ID) to each packet received, and per ingress port has a VLAN_PORT_FWD register to dictate which VLAN ID is used as forwarding VLAN ID to lookup VLAN table. In Q-in-Q application, outer VLAN ID is used as forwarding VLAN ID for both downlink and uplink ports as well as the inner VLAN ID is used in traditional VLAN application. Inner VLAN and outer VLAN decision flow will be described in following sections.

3.3.1 Port-based VLAN

Each ingress port supports a VLAN_PORT_PB_VLAN register to specify the inner, outer port-based VLAN ID and the packet format to apply port-based VLAN configuration.

Table 3: VLAN_PORT_PB_VLAN Register

Field Name	Bits	Description
OPVID	12	Outer port-based VLAN ID.
OPVID_FMT	2	Apply outer port-based VLAN ID on: 2'b00: outer untagged and outer priority tagged packet 2'b01: outer untagged packet 2'b10: all packet(outer untagged, outer priority-tagged, outer tagged) 2'b11: reserved
IPVID	12	Inner port-based VLAN ID.
IPVID_FMT	2	Apply inner port-based VLAN ID on: 2'b00: inner untagged and inner priority tagged packet 2'b01: inner untagged packet 2'b10: all packet(inner untagged, inner priority-tagged, inner tagged) 2'b11: reserved

API REFERENCE

```
rtk_vlan_portPvidMode_get(uint32 unit, rtk_port_t port, rtk_vlan_pbVlan_mode_t *pMode);
rtk_vlan_portPvidMode_set(uint32 unit, rtk_port_t port, rtk_vlan_pbVlan_mode_t mode);
rtk_vlan_portOuterPvidMode_get(uint32 unit, rtk_port_t port, rtk_vlan_pbVlan_mode_t *pMode);
rtk_vlan_portOuterPvidMode_set(uint32 unit, rtk_port_t port, rtk_vlan_pbVlan_mode_t mode);
rtk_vlan_portPvid_get(uint32 unit, rtk_port_t port, uint32 *pPvid);
rtk_vlan_portPvid_set(uint32 unit, rtk_port_t port, uint32 pvid);
rtk_vlan_portOuterPvid_get(uint32 unit, rtk_port_t port, rtk_vlan_t *pPvid);
rtk_vlan_portOuterPvid_set(uint32 unit, rtk_port_t port, rtk_vlan_t pvid);
```

3.3.2 Protocol-and-Port-based VLAN

The feature bases on the Frame Type and EtherType of the packet to assign the VLAN specified by each ingress port. Protocol-and-port-based VLAN of the device only assigns inner VLAN and doesn't support to assign outer VLAN. The Protocol-and-port-based VLAN ID applies to all the packet except the packet with inner tag (VID != 0). And the Protocol-and-port-based VLAN priority only applies to the packet which is both inner and outer untag.

The device supports global eight protocol-and-port-based VLAN configurations (VLAN_PPB_VLAN_VAL register), each configuration can specify the Frame Type and EtherType. Three frame types Ethernet, LLC_SNAP, LLC_Other are supported and the flow to determine the frame type is illustrated as below:

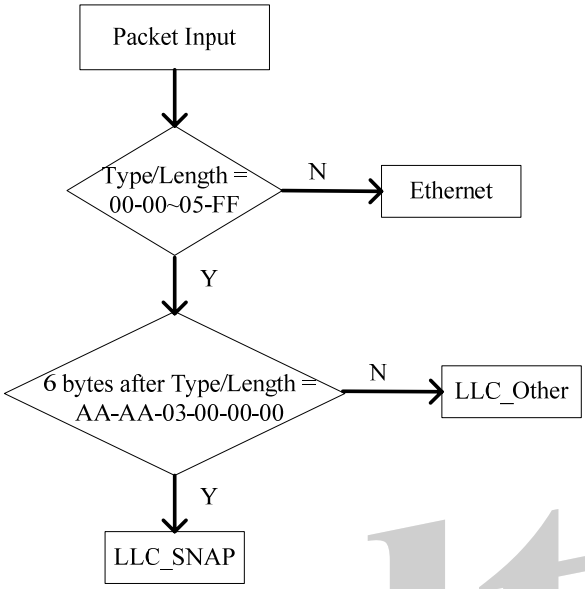


Figure 5: Frame Type Decision Flow

The EtherType field position of different packet encapsulation is illustrated as below:

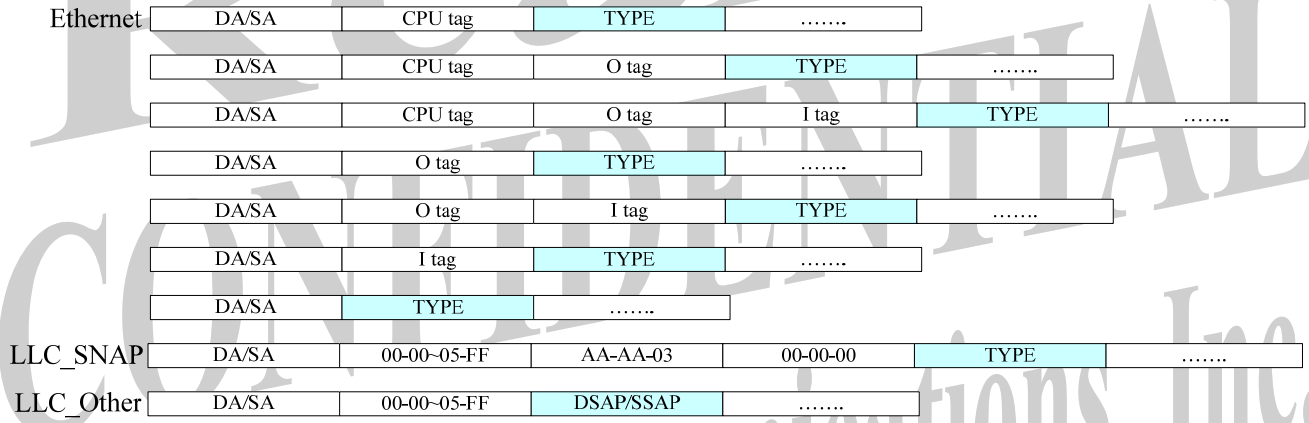


Figure 6: EtherType Field Position

Each ingress port supports eight protocol-and-port-based VLAN configurations (VLAN_PORT_PPB_VLAN register) mapping to global eight configurations, therefore, each port can have different VLAN ID and Priority assignment for the same protocol. Each configuration has a valid bit to enable the specific protocol-and-port-based VLAN configuration. The configuration with lowest index is used if multiple configurations are hit.

Table 4: VLAN_PORT_PPB_VLAN Register

Field Name	Bits	Description
VALID	1	Valid bit.
PPB_VID	12	Protocol-and-Port-based VLAN ID.
PPB_PRI	3	Protocol-and-Port-based VLAN Priority.

API REFERENCE

```
rtk_vlan_protoGroup_get(uint32 unit, uint32 protoGroup_idx, rtk_vlan_protoGroup_t *pProtoGroup);
rtk_vlan_protoGroup_set(uint32 unit, uint32 protoGroup_idx, rtk_vlan_protoGroup_t *pProtoGroup);
rtk_vlan_portProtoVlan_get(
    uint32                  unit,
    rtk_port_t              port,
```

```
    uint32                  protoGroup_idx,
    rtk_vlan_protoVlanCfg_t *pVlan_cfg);
rtk_vlan_portProtoVlan_set(
    uint32                  unit,
    rtk_port_t              port,
    uint32                  protoGroup_idx,
    rtk_vlan_protoVlanCfg_t *pVlan_cfg);
```

3.3.3 Ingress VLAN Translation Table

The device provides an ingress VLAN translation table which contains 1K entries to support upstream VLAN translation and Q-in-Q applications. Ingress VLAN translation table is used for VLAN translation upstream traffic and the VLAN assigned by ingress VLAN translation table is used as forwarding VLAN to forward the traffic and encapsulated to the upstream packet.
It can be used to support the 1:1/N:1 VLAN translation and Q-in-Q **upstream** models listed below but not limited to:

1:1 VLAN Translation/Q-in-Q Upstream Models
C → C'
C → S
C → S+C

N:1 VLAN Aggregation/Q-in-Q Upstream Models
C Range → C'
C Range → S
C Range → S+C

Ingress VLAN translation table is shown as below:

Table 5: Ingress VLAN Translation Table

Field Name	Bits	Description
Comparing Key		
VALID	1	Valid bit.
IVID	12	Inner VLAN ID. Configure the field for C → C'/ C → S/ C → S+C applications. The field should not be used with field IVID_RANGE_CHK.
IVID_RANGE_CHK	32	Inner VLAN ID Range Check Result (Refer to VLAN Range Check section for the detail). Each bit corresponds to a range check comparison result. Configure the field for C Range → C'/ C Range → S/ C Range → S+C applications. The field should not be used with field IVID.
IPRI	3	Inner User Priority.
PORT_ID	6	Ingress Port.
VALID_MASK	1	Valid bit mask.
IVID_MASK	12	Inner VLAN ID mask.
IVID_RANGE_CHK_MASK	32	Inner VLAN ID Range Check Result mask.
IPRI_MASK	3	Inner User Priority mask.
PORT_ID_MASK	6	Ingress Port mask.
Translation Data		
VLAN_SEL	1	1'b0: translated to a new inner VLAN (C → C' or C Range → C') 1'b1: translated to a outer VLAN (C → S or C → S+C or C Range → S or C Range → S+C)
VID_SHIFT	1	1'b0: translate the packet's VID to NEW_VID 1'b1: shift the packet's VID with NEW_VID
NEW_VID	12	VID_SHIFT = 0, NEW_VID will be the translated VID. VID_SHIFT = 1, NEW_VID is the offset value for translation. That is,

		translated VID = Packet's IVID + NEW_VID. Note1: VLAN_SEL = 0, translated VID represents translated I-VID. VLAN_SEL = 1, translated VID represents translated O-VID.
PRI_ASSIGN	1	1'b0: do not assign the priority. 1'b1: assign the priority.
NEW_PRI	3	VLAN_SEL = 0, replace inner user priority with NEW_PRI. And NEW_PRI is then remapped by inner 1P priority remapping table later. VLAN_SEL = 1, NEW_PRI is encapsulated to outer VLAN tag. The field is used if PRI_ASSIGN=1.
ITAG_STS	2	Assign egress inner VLAN tag status. 2'b00: inner untag. 2'b01: inner tag. 2'b10~2'b11: do not assign.
OTAG_STS	2	Assign egress outer VLAN tag status. 2'b00: outer untag. 2'b01: outer tag. 2'b10~2'b11: do not assign.
TPID_ASSIGN	1	1'b0: do not assign the TPID. 1'b1: assign the TPID and the TPID is indexed by TPID_INDEX.
TPID_IDX	2	The field is used if TPID_ASSIGN=1. VLAN_SEL = 0, index to inner TPID list. VLAN_SEL = 1, index to outer TPID list.

The comparing keys are implemented by TCAM, so each field has a corresponding mask which can mask off the comparing key. The comparing keys are used to qualify the packet while the translation data are used for doing ingress VLAN translation. In addition to translate to a new VID, priority and inner/outer VLAN tag status and TPID can also be assigned.
The ingress VLAN translation table takes effect for inner tag or inner priority-tag packet only and if there are multiple entries hit, the entry with lowest index is used.

API REFERENCE

```
rtk_vlan_igrVlanCnvtBlkMode_get(uint32 unit, uint32 blk_idx, rtk_vlan_igrVlanCnvtBlk_mode_t *pMode);
rtk_vlan_igrVlanCnvtBlkMode_set(uint32 unit, uint32 blk_idx, rtk_vlan_igrVlanCnvtBlk_mode_t mode);
rtk_vlan_igrVlanCnvtEntry_get(uint32 unit, uint32 index, rtk_vlan_igrVlanCnvtEntry_t *pData);
rtk_vlan_igrVlanCnvtEntry_set(uint32 unit, uint32 index, rtk_vlan_igrVlanCnvtEntry_t *pData);
rtk_vlan_igrVlanCnvtEntry_delAll(uint32 unit);
```

3.3.4 MAC-based VLAN

MAC-based VLAN and IP Subnet-based VLAN are supported by ingress VLAN translation table too. The device supports 1K ingress VLAN translation entries which are divided into 4 blocks. Each block can specify its working mode by setting VLAN_IGR_CNVT_BLK_CTRL.BLK_MODE field. If the block mode is configured to MAC-based VLAN, the table format is transformed to MAC-based VLAN table.

Table 6: MAC-based VLAN Table

Field Name	Bits	Description
Comparing Key		
VALID	1	Valid bit.
SMAC	48	Source MAC Address.
PORT_ID	6	Ingress Port.
VALID_MASK	1	Valid bit mask.
SMAC_MASK	48	Source MAC Address mask.

PORT_ID_MASK	6	Ingress Port mask.
Data		
NEW_VID	12	MAC-based VLAN ID.
NEW_PRI	3	MAC-based VLAN priority. The priority only applies to inner untag packet.

The MAC-based table takes effect for inner untag and inner priority-tag packet only and if there are multiple entries hit, the entry with lowest index is used.

API REFERENCE

```
rtk_vlan_igrVlanCnvtBlkMode_get(uint32 unit, uint32 blk_idx, rtk_vlan_igrVlanCnvtBlk_mode_t *pMode);
rtk_vlan_igrVlanCnvtBlkMode_set(uint32 unit, uint32 blk_idx, rtk_vlan_igrVlanCnvtBlk_mode_t mode);
rtk_vlan_macBasedVlan_get(
    uint32          unit,
    uint32          index,
    uint32          *pValid,
    rtk_mac_t       *pSmac,
    rtk_vlan_t      *pVid,
    rtk_pri_t       *pPriority);
rtk_vlan_macBasedVlan_set(
    uint32          unit,
    uint32          index,
    uint32          valid,
    rtk_mac_t       *pSmac,
    rtk_vlan_t      vid,
    rtk_pri_t       priority);
```

3.3.5 IP Subnet-based VLAN

If the ingress VLAN translation block mode is configured to IP Subnet-based VLAN, the table format is transformed to IP Subnet-based VLAN table.

Table 7: IP Subnet-based VLAN Table

Field Name	Bits	Description
Comparing Key		
VALID	1	Valid bit.
SIP	32	Source IP Address.
PORT_ID	6	Ingress Port.
VALID_MASK	1	Valid bit mask.
SIP_MASK	32	Source IP Address mask.
PORT_ID_MASK	6	Ingress Port mask.
Data		
NEW_VID	12	IP Subnet-based VLAN ID.
NEW_PRI	3	IP Subnet-based VLAN priority. The priority only applies to inner untag packet.

The IP Subnet-based table takes effect for inner untag and inner priority-tag packet only and if there are multiple entries hit, the entry with lowest index is used.

API REFERENCE

```
rtk_vlan_igrVlanCnvtBlkMode_get(uint32 unit, uint32 blk_idx, rtk_vlan_igrVlanCnvtBlk_mode_t *pMode);
rtk_vlan_igrVlanCnvtBlkMode_set(uint32 unit, uint32 blk_idx, rtk_vlan_igrVlanCnvtBlk_mode_t mode);
rtk_vlan_ipSubnetBasedVlan_get(
```

```
    uint32          unit,
    uint32          index,
    uint32          *pValid,
    ipaddr_t        *pSip,
    ipaddr_t        *pSip_mask,
    rtk_vlan_t      *pVid,
    rtk_pri_t       *pPriority);
rtk_vlan_ipSubnetBasedVlan_set(
    uint32          unit,
    uint32          index,
    uint32          valid,
    ipaddr_t        sip,
    ipaddr_t        sip_mask,
    rtk_vlan_t      vid,
    rtk_pri_t       priority);
```

3.3.6 Forwarding VLAN Decision

The device assigns two VLAN IDs (inner and outer VLAN ID) to each packet received, and one of them is chose as forwarding VLAN ID. Forwarding VLAN ID is used to lookup VLAN table and doing VLAN ingress/egress filtering. The device per ingress port supports a VLAN_PORT_FWD register to dictate which VLAN ID is used as learning and forwarding VLAN ID.

The flow to determine the ingress inner and outer VLAN ID is illustrated below.

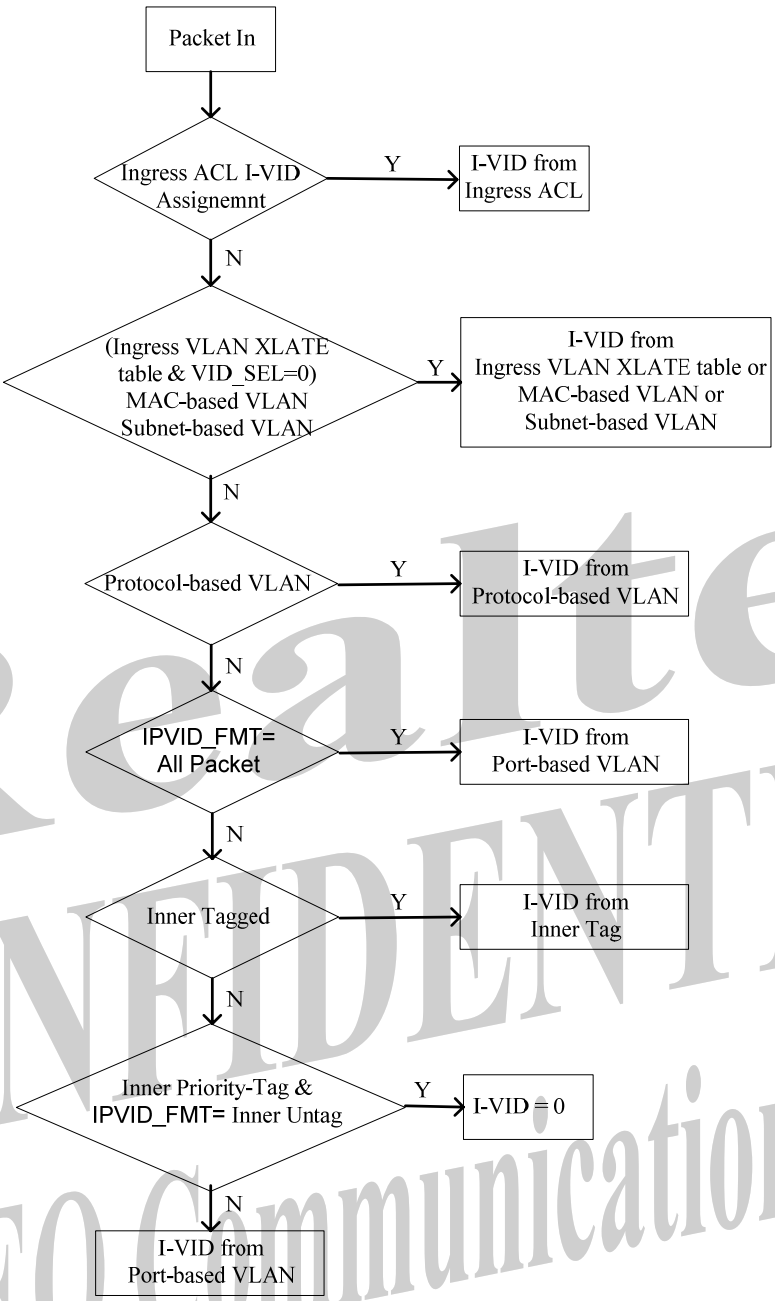


Figure 7: Ingress Inner VLAN ID Decision Flow

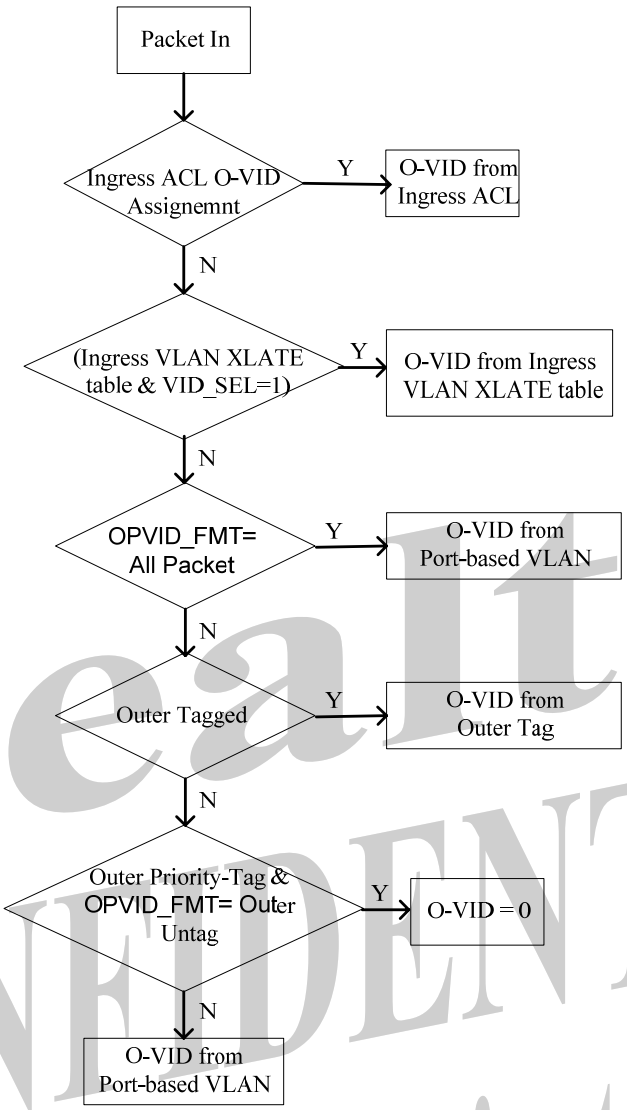


Figure 8: Ingress Outer VLAN ID Decision Flow

Because ingress VLAN translation table can also be used to support MAC-based VLAN and IP Subnet-based VLAN per block based, if a packet hit the ingress VLAN translation table, MAC-based VLAN and IP Subnet-based VLAN concurrently, the entry with global lowest index is taken.

API REFERENCE

```
rtk_l2_vlanMode_get(uint32 unit, rtk_port_t port, rtk_l2_vlanMode_t *pVlanMode);
rtk_l2_vlanMode_set(uint32 unit, rtk_port_t port, rtk_l2_vlanMode_t vlanMode);
```

3.4 VLAN Lookup

When the forwarding VLAN is determined by VLAN decision flow, it is used as index to lookup the VLAN table for retrieving the member ports and untag member ports.

3.4.1 VLAN Table

The device supports a 4K (0-4095) entries VLAN table. The format of VLAN table entry is shown as below:

Table 8: VLAN Table

Field Name	Bits	Description
MBR	53	Member port.
UNTAG_MBR	53	Untag member port.
FID_MSTI	8	Filtering Database ID/Multiple Spanning Tree Instance ID.
L2_HASH_KEY_UC	1	Specify the VLAN learning mode for L2 unicast and broadcast traffic. 1'b0: IVL. VID is used for learning MAC address. 1'b1: SVL. FID_MSTI is used for learning MAC address.
L2_HASH_KEY_MC	1	Specify the VLAN learning mode for L2 and IP multicast traffic. 1'b0: IVL. VID is used for learning MAC address. 1'b1: SVL. FID_MSTI is used for learning MAC address.
VLAN_PROFILE	3	VLAN profile index.

MBR — Define the VLAN member port. VLAN member port is utilized by VLAN ingress and egress filtering.

UNTAG_MBR — Define the VLAN untag member port. The VLAN tag status of an outgoing packet is determined by both UNTAG_MBR setting and egress port tag status configuration. Please refer to Egress VLAN Tagging Status section for the detail.

FID_MSTI — The field represents both Filtering Database ID and Multiple Spanning Tree Instance ID. FID is used for SVL while MSTI is used for Spanning Tree ingress/egress filtering. There are 256 FID/MSTI supported by the device.

L2_HASH_KEY_UC — Per VLAN specify the VLAN learning mode for L2 unicast and broadcast traffic. IVL/SVL mixed mode can be supported through configuring the field.

L2_HASH_KEY_MC — Per VLAN specify the VLAN learning mode for L2 and IP multicast traffic. IVL/SVL mixed mode can be supported through configuring the field.

VLAN_PROFILE — Please refer to VLAN Profile section for the detail.

There is a configuration VLAN_CTRL.EXCPT to specify the forwarding behavior (forward/drop/trap) for the packet which VLAN member is 0 or the forwarding VLAN ID is 4095.

API REFERENCE

```
rtk_vlan_create(uint32 unit, rtk_vlan_t vid);
rtk_vlan_destroy(uint32 unit, rtk_vlan_t vid);
rtk_vlan_destroyAll(uint32 unit, uint32 restore_default_vlan);
rtk_vlan_port_add(uint32 unit, rtk_vlan_t vid, rtk_port_t port, uint32 is_untag);
rtk_vlan_port_del(uint32 unit, rtk_vlan_t vid, rtk_port_t port);
rtk_vlan_port_get(
    uint32          unit,
    rtk_vlan_t      vid,
    rtk_portmask_t *pMember_portmask,
    rtk_portmask_t *pUntag_portmask);
rtk_vlan_port_set(
    uint32          unit,
    rtk_vlan_t      vid,
    rtk_portmask_t *pMember_portmask,
    rtk_portmask_t *pUntag_portmask);
rtk_vlan_stg_get(uint32 unit, rtk_vlan_t vid, rtk_stg_t *pStg);
rtk_vlan_stg_set(uint32 unit, rtk_vlan_t vid, rtk_stg_t stg);
rtk_vlan_l2UcastLookupMode_get(uint32 unit, rtk_vlan_t vid, rtk_l2_ucastLookupMode_t *pMode);
rtk_vlan_l2UcastLookupMode_set(uint32 unit, rtk_vlan_t vid, rtk_l2_ucastLookupMode_t mode);
rtk_vlan_l2McastLookupMode_get(uint32 unit, rtk_vlan_t vid, rtk_l2_mcastLookupMode_t *pMode);
rtk_vlan_l2McastLookupMode_set(uint32 unit, rtk_vlan_t vid, rtk_l2_mcastLookupMode_t mode);
rtk_vlan_profileIdx_get(uint32 unit, rtk_vlan_t vid, uint32 *pIdx);
rtk_vlan_profileIdx_set(uint32 unit, rtk_vlan_t vid, uint32 idx);
rtk_vlan_except_get(uint32 unit, rtk_action_t *pAction);
rtk_vlan_except_set(uint32 unit, rtk_action_t action);
```

3.4.2 VLAN Profile

The device supports 8 VLAN profiles. VLAN profile is used to specify the SA learning behavior and unknown multicast flooding port mask. VLAN profile structure is shown as below:

Table 9: VLAN Profile

Field Name	Bits	Description
L2_LRN_EN	1	Enable MAC address learning.
L2_UNKN_MC_FLD_PMSK	12	An index points to multicast table for retrieving unknown L2 multicast flooding port mask.
IP4_UNKN_MC_FLD_PMSK	12	An index points to multicast table for retrieving unknown IPv4 multicast flooding port mask.
IP6_UNKN_MC_FLD_PMSK	12	An index points to multicast table for retrieving unknown IPv6 multicast flooding port mask.

Because each VLAN maps to a VLAN profile, disable MAC address learning or have different flooding port mask on specific VLAN is practicable.

API REFERENCE

```
rtk_vlan_profile_get(uint32 unit, uint32 idx, rtk_vlan_profile_t *pProfile);
rtk_vlan_profile_set(uint32 unit, uint32 idx, rtk_vlan_profile_t *pProfile);
```

3.5 VLAN Filtering

A packet is examined by VLAN ingress filtering when it is received and examined by VLAN egress filtering before it is transmitted.

3.5.1 VLAN Ingress Filtering

A packet is asserted by VLAN ingress filtering if its source port is not in the member ports. The device per ingress port supports a VLAN_PORT_IGR_FLTR register to handle the packet.

Table 10: VLAN_PORT_IGR_FLTR Register

Field Name	Bits	Description
IGR_FLTR_ACT	2	VLAN ingress filtering action. 2'b00: forward 2'b01: drop 2'b10: trap 2'b11: reserved

API REFERENCE

```
rtk_vlan_portIgrFilter_get(uint32 unit, rtk_port_t port, rtk_vlan_ifilter_t *pIgr_filter);
rtk_vlan_portIgrFilter_set(uint32 unit, rtk_port_t port, rtk_vlan_ifilter_t igr_filter);
```

3.5.2 VLAN Egress Filtering

A packet should only be forwarded to the VLAN where it came from. This is achieved by VLAN egress filtering. However, the device per egress port supports a VLAN_PORT_EGR_FLTR register to disable the VLAN egress filtering. When the VLAN egress filtering is disabled at a certain port, all the traffic (unicast/multicast/broadcast)

forwarded to that port will bypass the VLAN egress filtering.
When VLAN egress filtering is enabled, the device globally supports a VLAN_CTRL.LKY field to disable the VLAN egress filtering for known L2/IPv4/IPv6 multicast traffic to across VLAN. The VLAN Leaky configuration (VLAN_CTRL.LKY) doesn't apply to L2 unicast and broadcast traffic.

API REFERENCE

```
rtk_vlan_portEgrFilterEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);
rtk_vlan_portEgrFilterEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
rtk_vlan_mcastLeakyEnable_get(uint32 unit, rtk_enable_t *pLeaky);
rtk_vlan_mcastLeakyEnable_set(uint32 unit, rtk_enable_t leaky);
```

3.6 Egress VLAN Assignment

In addition to assign two ingress VLAN IDs (inner and outer VLAN ID) to each packet received, the device also determines two egress VLAN IDs for the packet. Unlike ingress VLAN ID, the egress VLAN ID doesn't participate VLAN forwarding and VLAN ingress/egress filtering processing but encapsulates to the packet before transmitting.
Each packet is determined two egress VLAN IDs before the egress tag status decision is made. They are used for encapsulation only if the transmitting packet carries VLAN tags. Regarding the egress tag status decision flow, please refer to Egress VLAN Tagging Status section.

3.6.1 Egress VLAN Translation Table

The device provides an egress VLAN translation table which contains 1K entries to support downstream VLAN translation and Q-in-Q applications. Egress VLAN translation table is used for VLAN translation downstream traffic and the VLAN assigned by egress VLAN translation table is encapsulated to the downstream packet.
It can be used to support the 1:1 VLAN translation and Q-in-Q **downstream** models listed below but not limited to:

```
1:1 VLAN Translation/Q-in-Q Downstream Models
C' → C
S → C
```

A configuration register VLAN_EGR_CNVT_CTRL is provided to specify the VLAN source and specify whether translate the double tag packet.

Table 11: VLAN_EGR_CNVT_CTRL Register

Field Name	Bits	Description
VID_SRC	1	Egress VLAN translation VLAN ID source. 1'b0: inner VLAN (C' → C) 1'b1: outer VLAN (S → C)
DBL_TAG_EN	1	Apply egress VLAN translation to double (outer + inner) tag packet. 1'b0: disable 1'b1: enable

Egress VLAN translation table is shown as below:

Table 12: Egress VLAN Translation Table

Field Name	Bits	Description
Comparing Key		
VALID	1	Valid bit.
VID	12	VLAN ID either from outer tag or inner tag, refer to VLAN_EGR_CNVT_CTRL.VID_SRC. The field should not be used with field VID_RANGE_CHK.
VID_RANGE_CHK	32	VLAN ID Range Check Result (Refer to VLAN Range Check section for the detail). Each bit corresponds to a range check comparison result.

		The field should not be used with field VID.
PRI	3	Inner or Outer User Priority, refer to VLAN_EGR_CNVT_CTRL.VID_SRC.
PORT_ID	6	Egress Port.
VALID_MASK	1	Valid bit mask.
VID_MASK	12	VLAN ID mask.
VID_RANGE_CHK_MASK	32	VLAN ID Range Check Result mask.
PRI_MASK	3	User Priority mask.
PORT_ID_MASK	6	Egress Port mask.
Translation Data		
VID_SHIFT	1	1'b0: translate the packet's VID to NEW_VID 1'b1: shift the packet's VID with NEW_VID
NEW_VID	12	VID_SHIFT = 0, NEW_VID will be the translated inner VID. VID_SHIFT = 1, NEW_VID is the offset value for translation. That is, translated inner VID = Packet's VID + NEW_VID.
PRI_ASSIGN	1	1'b0: do not assign the priority. 1'b1: assign the priority.
NEW_PRI	3	NEW_PRI is encapsulated to inner VLAN tag and supersede the egress inner tag remarking. It doesn't participate ingress priority decision. The field is used if PRI_ASSIGN=1.
TPID_ASSIGN	1	1'b0: do not assign the inner TPID. 1'b1: assign the inner TPID and the TPID is indexed by TPID_INDEX.
TPID_IDX	2	Index to inner TPID list. The field is used if TPID_ASSIGN=1.

The comparing keys are implemented by TCAM, so each field has a corresponding mask which can mask off the comparing key. The comparing keys are used to qualify the packet while the translation data are used for doing egress VLAN translation. In addition to translate to a new inner VID, inner priority and inner TPID can also be assigned. As for egress tag status, packet hit egress VLAN translation table is forced to be inner tag.
The egress VLAN translation table takes effect for tagged (exclude priority-tag) packet only and if there are multiple entries hit, the entry with lowest index is used.

API REFERENCE

```
rtk_vlan_egrVlanCnvtVidSource_get(uint32 unit, rtk_l2_vlanMode_t *pSrc);
rtk_vlan_egrVlanCnvtVidSource_set(uint32 unit, rtk_l2_vlanMode_t src);
rtk_vlan_egrVlanCnvtDblTagEnable_get(uint32 unit, rtk_enable_t *pEnable);
rtk_vlan_egrVlanCnvtDblTagEnable_set(uint32 unit, rtk_enable_t enable);
rtk_vlan_egrVlanCnvtEntry_get(uint32 unit, uint32 index, rtk_vlan_egrVlanCnvtEntry_t *pData);
rtk_vlan_egrVlanCnvtEntry_set(uint32 unit, uint32 index, rtk_vlan_egrVlanCnvtEntry_t *pData);
rtk_vlan_egrVlanCnvtEntry_delAll(uint32 unit);
```

3.6.2 MAC-based N:1 VLAN Aggregation

MAC-based N:1 VLAN aggregation upstream is done by ingress VLAN translation table. For downstream, the device per egress port supports a VLAN_PORT_NTO1_AGGR register which enables translating the aggregation VLAN ID back to its original inner VLAN ID. **The downstream N:1 VLAN translation only applies to known unicast packet.**

API REFERENCE

```
rtk_vlan_portVlanAggrEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);
rtk_vlan_portVlanAggrEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
```

3.6.3 Egress VLAN Decision

The flow to determine the egress inner and outer VLAN ID is illustrated below.

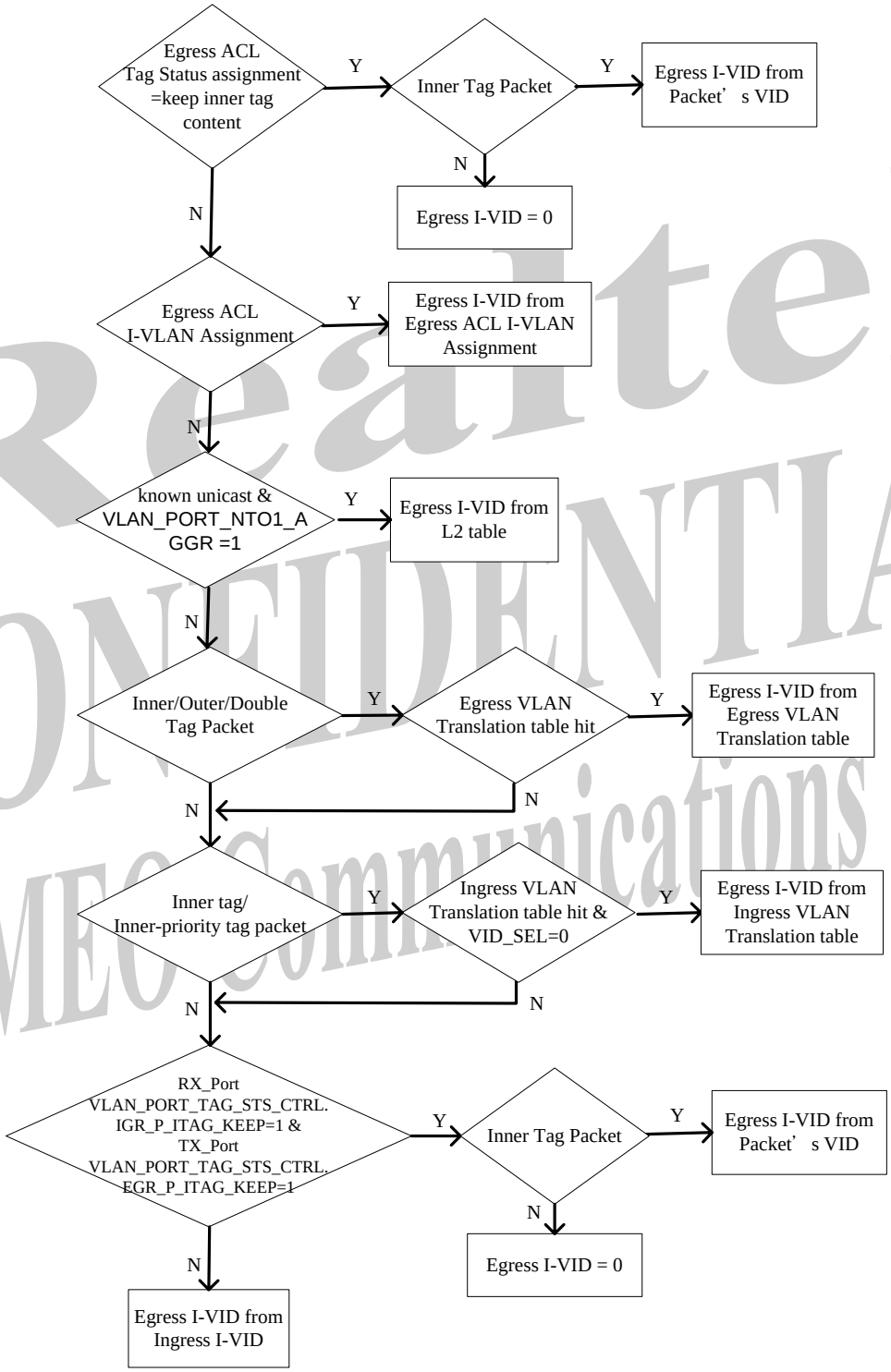


Figure 9: Egress Inner VLAN ID Decision Flow

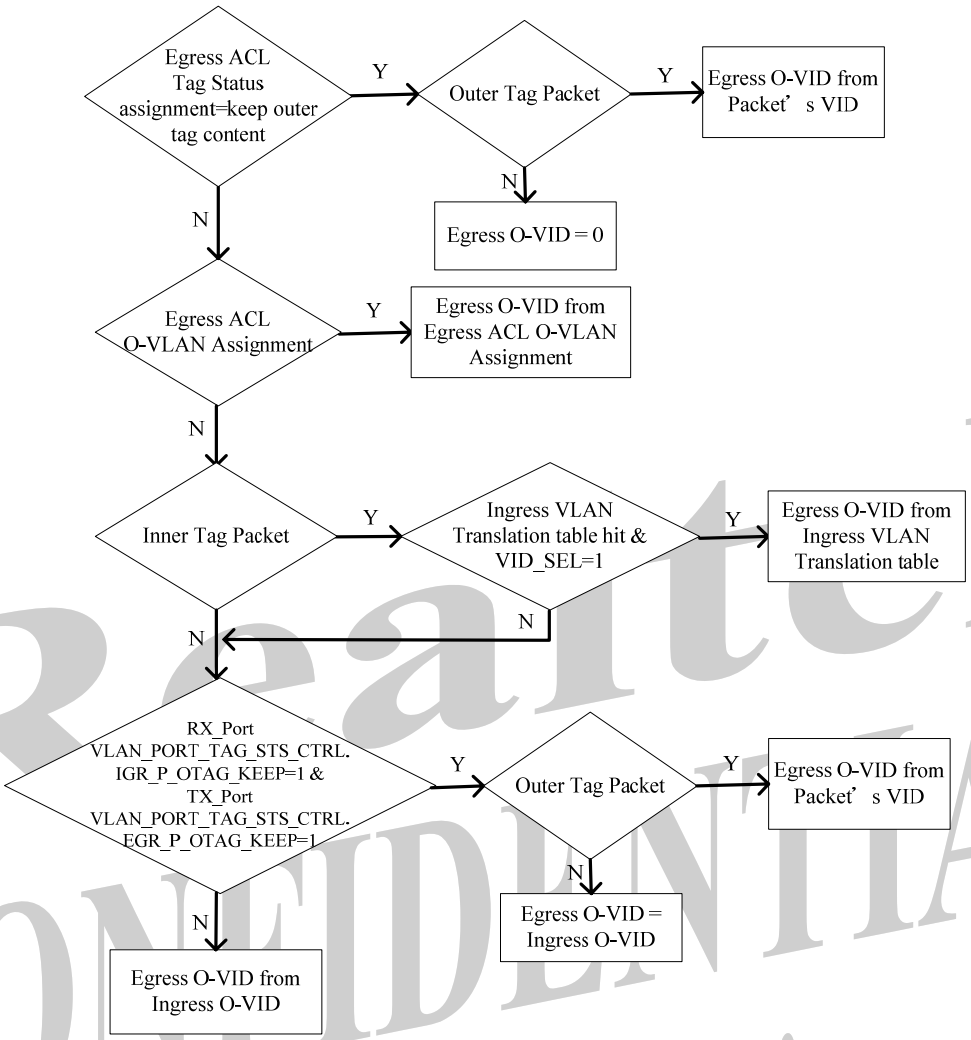


Figure 10: Egress Outer VLAN ID Decision Flow

The section only describes the egress VLAN ID decision flow. Regarding the priority and CFI field decision flow, please refer to Egress Remarking Developer's Guide.

3.7 Egress VLAN Tag Manipulation

The device determines egress inner and outer tag status for a packet before transmitting. Double VLAN tag insertion, remove, keep operations are supported for flexible Q-in-Q VLAN application. In addition to Port-based and VLAN-based tag status assignment, flow-based is also supported by Egress ACL.

3.7.1 Egress VLAN Tagging Status

The device per port supports a VLAN_PORT_TAG_STS_CTRL register to specify the VLAN tag processing behavior.

Table 13: VLAN_PORT_TAG_STS_CTRL Register

Field Name	Bits	Description
IGR_P_ITAG_KEEP	1	Per ingress port specify the egress inner VLAN tag processing mode. 1'b0: normal (follow VLAN untag set and ITAG_STS configuration) 1'b1: keep (keep VLAN tag status and content)
IGR_P_OTAG_KEEP	1	Per ingress port specify the egress outer VLAN tag processing mode.

		1'b0: normal (follow VLAN untag set and OTAG_STS configuration) 1'b1: keep (keep VLAN tag status and content)
EGR_P_ITAG_KEEP	1	Per engress port specify the egress inner VLAN tag processing mode. 1'b0: normal (follow VLAN untag set and ITAG_STS configuration) 1'b1: keep (keep VLAN tag status and content)
EGR_P_OTAG_KEEP	1	Per engress port specify the egress outer VLAN tag processing mode. 1'b0: normal (follow VLAN untag set and OTAG_STS configuration) 1'b1: keep (keep VLAN tag status and content)
ITAG_STS	2	Inner VLAN tag status. 2'b00: inner untag 2'b01: inner tag 2'b10: inner priority-tag 2'b11: reserved
OTAG_STS	2	Outer VLAN tag status. 2'b00: inner untag 2'b01: inner tag 2'b10: inner priority-tag 2'b11: reserved

The inner VLAN tag status of a packet is kept if the configuration IGR_P_ITAG_KEEP of packet's ingress port and the configuration EGR_P_ITAG_KEEP of packet's egress port are BOTH set to 'keep'.
The outer VLAN tag status of a packet is kept if the configuration IGR_P_OTAG_KEEP of packet's ingress port and the configuration EGR_P_OTAG_KEEP of packet's egress port are BOTH set to 'keep'.
If the VLAN tag status is determined to be 'keep', the tag format and its content (VID/Priority/CFI) are all kept. Keeping tag format means that untag in will be untag out, priority-tag in will be priority-tag out, tag in will be tag out.

If either IGR_P_ITAG_KEEP (IGR_P_OTAG_KEEP) of packet's ingress port or EGR_P_ITAG_KEEP (EGR_P_OTAG_KEEP) of packet's egress port is set to 'normal', the VLAN tag status is then determined by VLAN untag set and ITAG_STS (OTAG_STS) configuration of egress port. According to forwarding VLAN configuration (VLAN_PORT_FWD), the VLAN tag processing is different.
If forwarding VLAN is from inner VLAN, the packet is determined to be inner untag if its egress port is in the VLAN untag member, otherwise, the inner tag status is directly determined by ITAG_STS field. However, outer tag status is simply determined by OTAG_STS field.
If forwarding VLAN is from outer VLAN, the packet is determined to be outer untag if its egress port is in the VLAN untag member, otherwise, the outer tag status is directly determined by OTAG_STS field. However, inner tag status is simply determined by ITAG_STS field.

The flow to determine the egress inner and outer VLAN tag status is illustrated below.

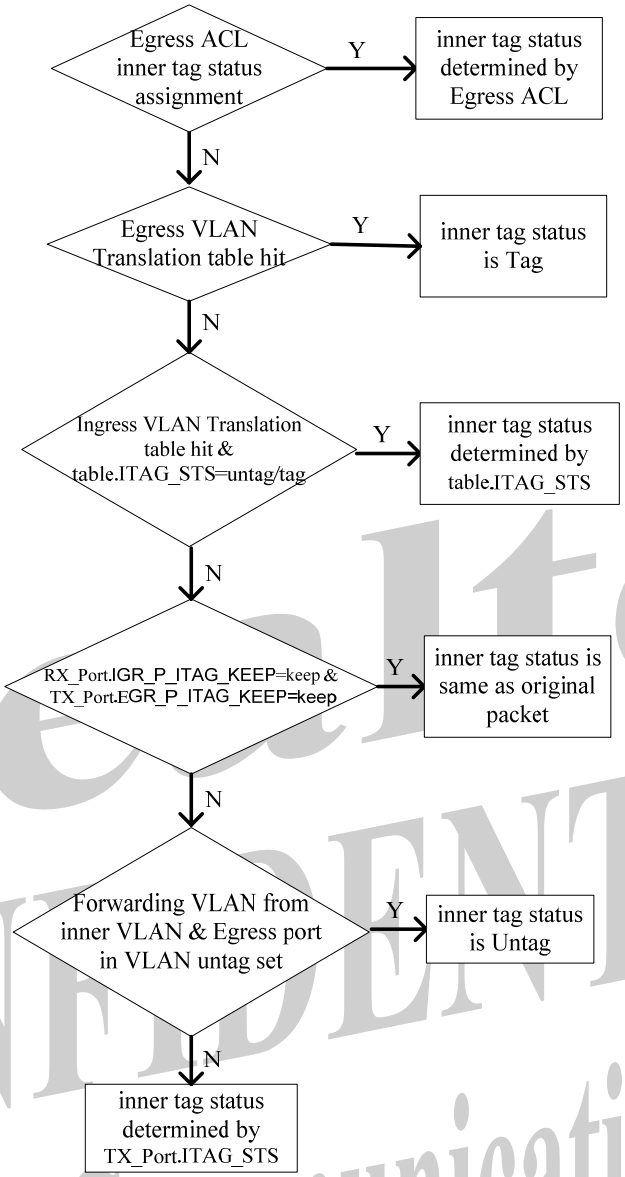


Figure 11: Egress Inner VLAN Tag Status Decision Flow

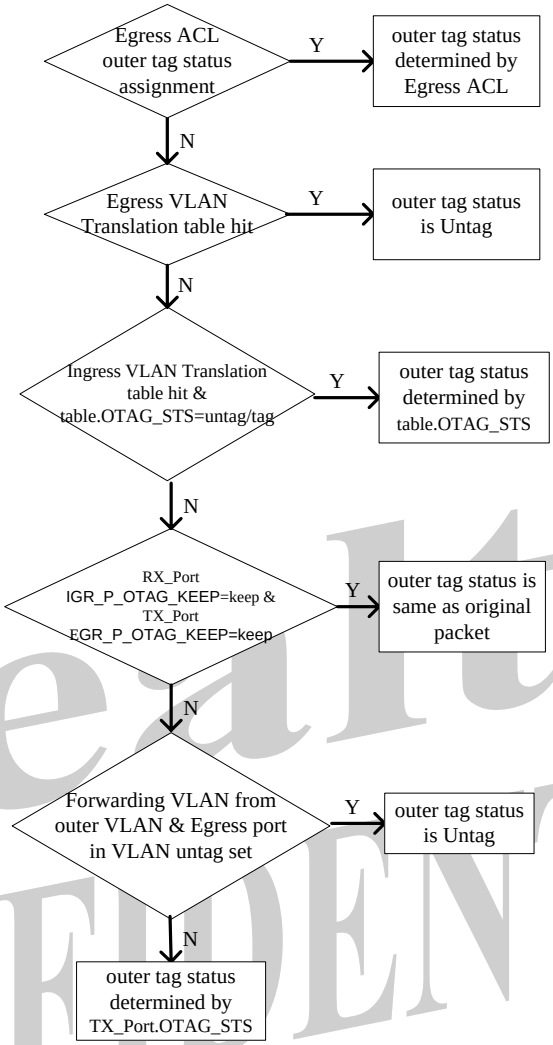


Figure 12: Egress Outer VLAN Tag Status Decision Flow

API REFERENCE

```
rtk_vlan_portIgrTagKeepEnable_get(
    uint32          unit,
    rtk_port_t      port,
    rtk_enable_t    *pKeepOuter,
    rtk_enable_t    *pKeepInner);
rtk_vlan_portIgrTagKeepEnable_set(
    uint32          unit,
    rtk_port_t      port,
    rtk_enable_t    keepOuter,
    rtk_enable_t    keepInner);
rtk_vlan_portEgrTagKeepEnable_get(
    uint32          unit,
    rtk_port_t      port,
    rtk_enable_t    *pKeepOuter,
    rtk_enable_t    *pKeepInner);
rtk_vlan_portEgrTagKeepEnable_set(
    uint32          unit,
    rtk_port_t      port,
    rtk_enable_t    keepOuter,
```

```
    rtk_enable_t       keepInner);
rtk_vlan_portEgrInnerTagSts_get(uint32 unit, rtk_port_t port, rtk_vlan_tagSts_t *pSts);
rtk_vlan_portEgrInnerTagSts_set(uint32 unit, rtk_port_t port, rtk_vlan_tagSts_t sts);
rtk_vlan_portEgrOuterTagSts_get(uint32 unit, rtk_port_t port, rtk_vlan_tagSts_t *pSts);
rtk_vlan_portEgrOuterTagSts_set(uint32 unit, rtk_port_t port, rtk_vlan_tagSts_t sts);
```

3.7.2 Egress VLAN Tagging TPID

Port-based, VLAN-based and flow-based VLAN TPID is supported. The device per egress port supports a VLAN_PORT_EGR_ITPID_CTRL.ITPID_IDX and VLAN_PORT_EGR_OTPID_CTRL.OTPID_IDX register field to specify the inner and outer TPID respectively. Please note instead specify the TPID value directly; the field specify the index to TPID list. The configuration doesn't take effect if received packet originally carries inner (outer) tag, that is, inner (outer) TPID is kept if received packet originally carries inner (outer) tag.
VLAN-based TPID is supported by ingress and egress VLAN translation table while flow-based TPID is supported by egress ACL. The flow to determine the egress inner and outer VLAN TPID is illustrated below.

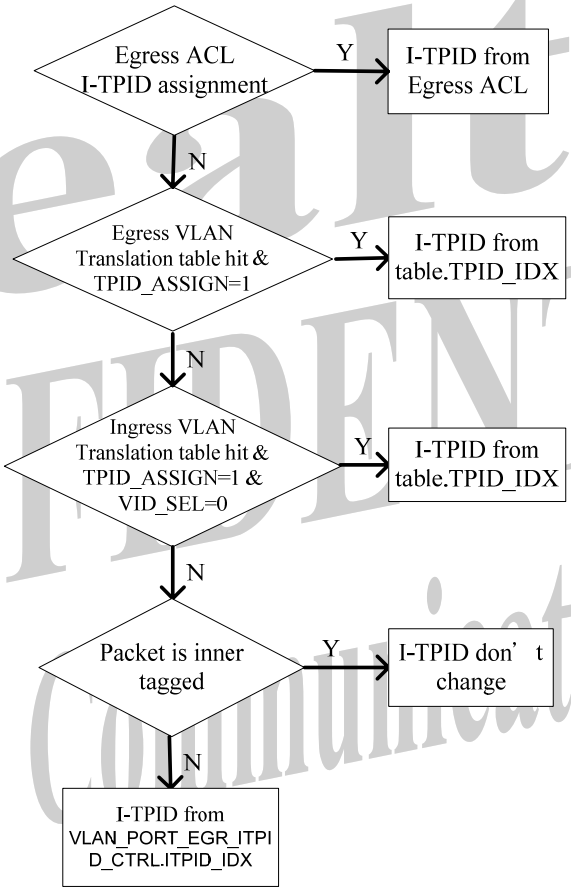


Figure 13: Egress Inner VLAN TPID Decision Flow

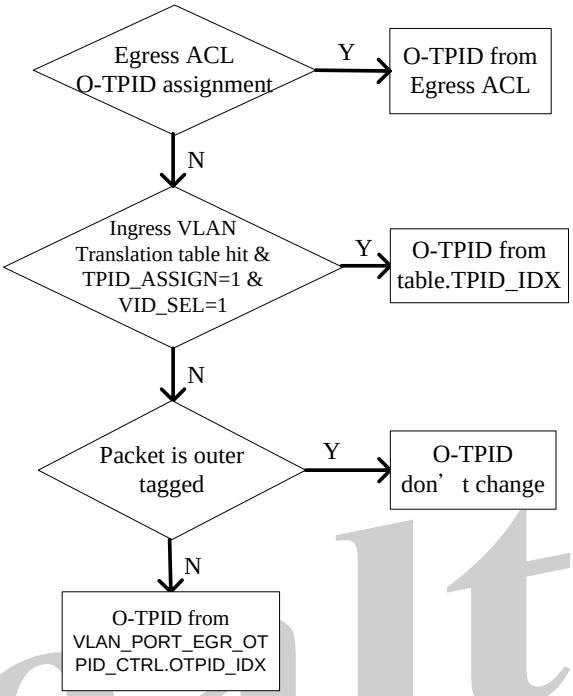


Figure 14: Egress Outer VLAN TPID Decision Flow

API REFERENCE

```
rtk_vlan_portEgrInnerTpid_get(uint32 unit, rtk_port_t port, uint32 *pTpid_idx);
rtk_vlan_portEgrInnerTpid_set(uint32 unit, rtk_port_t port, uint32 tpid_idx);
rtk_vlan_portEgrOuterTpid_get(uint32 unit, rtk_port_t port, uint32 *pTpid_idx);
rtk_vlan_portEgrOuterTpid_set(uint32 unit, rtk_port_t port, uint32 tpid_idx);
```

3.8 VLAN Translation

The device provides 1K ingress and 1K egress VLAN translation entries to support VLAN translation and flexible Q-in-Q applications. Ingress VLAN translation table is used for VLAN translation/Q-in-Q upstream traffic while egress VLAN translation table is for VLAN translation/Q-in-Q downstream traffic. The major difference between ingress and egress VLAN translation is that the VLAN assigned by ingress VLAN translation table is used as forwarding VLAN to forward the traffic and encapsulated to the upstream packet while the VLAN assigned by egress VLAN translation table just encapsulated to the downstream packet and do not participate the VLAN forwarding process.

3.8.1 VLAN Range Check

The device supports 32 VLAN range check configurations RNG_CHK_VID_CTRL which used for ingress VLAN translation table and ACL field type (Please refer to Ingress and Egress ACL Developer's Guide). The received packet is compared by 32 VLAN range check configurations concurrently and the 32-bit comparison result is then examined by the field IVID_RANGE_CHK of ingress VLAN translation table.

Table 14: RNG_CHK_VID_CTRL Register

Field Name	Bits	Description
VID_UPPER	12	VLAN range upper bound.
VID_LOWER	12	VLAN range lower bound.
REVERSE	1	Reverse comparison result.
TYPE	1	Specify the VLAN source.

	1'b0: inner VLAN 1'b1: outer VLAN

There are also 32 dedicated VLAN range check configurations RNG_CHK_VID_EGR_XLATE_CTRL supported for egress VLAN translation table. The received packet is compared by 32 VLAN range check configurations concurrently and the 32-bit comparison result is then examined by the field VID_RANGE_CHK of egress VLAN translation table.

Table 15: RNG_CHK_VID_EGR_XLATE_CTRL Register

Field Name	Bits	Description
VID_UPPER	12	VLAN range upper bound.
VID_LOWER	12	VLAN range lower bound.
REVERSE	1	Reverse comparison result.
TYPE	1	Specify the VLAN source. 1'b0: inner VLAN 1'b1: outer VLAN

The comparison result is valid if the specified VLAN source is available of the received packet and reverse operation only takes effect for valid comparison result.

API REFERENCE

```
rtk_acl_rangeCheckVid_get(uint32 unit, uint32 index, rtk_acl_rangeCheck_vid_t *pData);
rtk_acl_rangeCheckVid_set(uint32 unit, uint32 index, rtk_acl_rangeCheck_vid_t *pData);
rtk_vlan_egrVlanCnvtRangeCheckVid_get(
    uint32                                  unit,
    uint32                                  index,
    rtk_vlan_egrVlanCnvtRangeCheck_vid_t    *pData);
rtk_vlan_egrVlanCnvtRangeCheckVid_set(
    uint32                                  unit,
    uint32                                  index,
    rtk_vlan_egrVlanCnvtRangeCheck_vid_t    *pData);
```

3.8.2 Application Example

Example 1: VLAN 1:1 Mapping

Condition:
Packet from downlink port with inner VLAN 10, and translates to inner VLAN 100 before sending out uplink port.

```
rtk_vlan_igrVlanCnvtEntry_t    data ;

/* set ingress VLAN translation */
rtk_vlan_igrVlanCnvtBlkMode_set(unit, 0, CONVERSION_MODE_C2SC);

/* set customer uplink translation information (10->100) */
osal_memset(&data, 0, sizeof(rtk_vlan_igrVlanCnvtEntry_t));
data.vid = 10;
data.vid_cnvt_sel = 0;
data.inner_tag_sts = 1;
data.vid_new = 100;
data.valid = True;
rtk_vlan_igrVlanCnvtEntry_set(unit, 0, &data);

/* set customer downlink translation information (100->10) */
```

```
osal_memset(&data, 0, sizeof(rtk_vlan_igrVlanCnvtEntry_t));
data.vid = 100;
data.vid_cnvt_sel = 0;
data.vid_new = 10;
data.valid = True;
rtk_vlan_igrVlanCnvtEntry_set(unit, 1, &data);
```

Example 2: VLAN 1:1 Shift Mapping

Condition:
Packet from downlink port with inner VLAN 10, and translates to inner VLAN 210 before sending out uplink port.
Packet from downlink port with inner VLAN 11, and translates to inner VLAN 211 before sending out uplink port.

```
rtk_vlan_igrVlanCnvtEntry_t   data ;
rtk_acl_rangeCheck_vid_t     vRng;
int32                        rngIdx;

/* set ingress VLAN translation */
rtk_vlan_igrVlanCnvtBlkMode_set(unit, 0, CONVERSION_MODE_C2SC);

/* set customer uplink translation information (10->210, 11->211) */
rngIdx = 0;
osal_memset(&vRng, 0, sizeof(rtk_acl_rangeCheck_vid_t));
vRng. vid_lower_bound = 10;
vRng. vid_upper_bound = 11;
rtk_acl_rangeCheckVid_set(unit, rngIdx, &vRng);

osal_memset(&data, 0, sizeof(rtk_vlan_igrVlanCnvtEntry_t));
data.vid_ignore = True;
data.rngchk_result = data.rngchk_result_mask = (1 << rngIdx);
data.vid_cnvt_sel = 0;
data.inner_tag_sts = 1;
data.vid_new = 200;
data.vid_shift = True;
data.valid = True;
rtk_vlan_igrVlanCnvtEntry_set(unit, 0, &data);

/* set customer downlink translation information (210->10, 211->11) */
rngIdx = 1;
osal_memset(&vRng, 0, sizeof(rtk_acl_rangeCheck_vid_t));
vRng. vid_lower_bound = 210;
vRng. vid_upper_bound = 211;
rtk_acl_rangeCheckVid_set(unit, rngIdx, &vRng);

osal_memset(&data, 0, sizeof(rtk_vlan_igrVlanCnvtEntry_t));
data.vid_ignore = True;
data.rngchk_result = data.rngchk_result_mask = (1 << rngIdx);
data.vid_cnvt_sel = 0;
data.vid_new = 200;
data.vid_shift = True;
data.vid_shift_sel = 1;
data.valid = True;
rtk_vlan_igrVlanCnvtEntry_set(unit, 0, &data);
```

Example 3: VLAN N:1 Mapping

Condition:
Packet from downlink port with inner VLAN 10~20, and translates to outer VLAN 300 before sending out uplink port.

```
rtk_vlan_igrVlanCnvtEntry_t   data ;
rtk_acl_rangeCheck_vid_t     vRng;
int32                        rngIdx;

/* set ingress VLAN translation */
rtk_vlan_igrVlanCnvtBlkMode_set(unit, 0, CONVERSION_MODE_C2SC);

/* set customer uplink translation information (10~20->300) */
rngIdx = 0;
osal_memset(&vRng, 0, sizeof(rtk_acl_rangeCheck_vid_t));
vRng. vid_lower_bound = 10;
vRng. vid_upper_bound = 20;
rtk_acl_rangeCheckVid_set(unit, rngIdx, &vRng);

osal_memset(&data, 0, sizeof(rtk_vlan_igrVlanCnvtEntry_t));
data.vid_ignore = True;
data.rngchk_result = data.rngchk_result_mask = (1 << rngIdx);
data.vid_cnvt_sel = 1;
data.outer_tag_sts = 1;
data.vid_new = 300;
data.valid = True;
rtk_vlan_igrVlanCnvtEntry_set(unit, 0, &data);

/* set customer downlink translation information (strip outer tag) */
rtk_vlan_portVlanAggrEnable_set(unit, uplinkPort, ENABLED);
```

4 Layer 2 Switching

The Layer 2 packet forwarding based on Layer 2 header is basis of a switch. The device provides a Layer 2 table with 16K entries to record unicast/multicast MAC and its port mapping. For multicast forwarding, it has multiple destination ports, the device provides a 4K entry L2 PortMask table for storing port mask (This table is also referred by VLAN profile and ACL).

The device also provides Static, Dynamic Port Move and Port Move Forbidding mechanism to handle port movement for security applications. When an incoming packet lookup miss, system also provides L2 Unicast Lookup Miss, L2 Multicast Lookup Miss IPv4 Multicast Lookup Miss and IPv6 Multicast Lookup Miss operations, and is able to define the flooding domain.

For security concern, the device also supports MAC Constraint function to limit system Layer 2 address learning count, per-port based learning count, and another 32 entries VLAN-and-Port based address learning count.

In management switch, the Layer 2 entry aging-out/new learn/port move may be concerned by management software. The L2 Notification can acknowledge CPU with above events in Notification buffer in memory.
The configurable Layer 2 entry aging out time ranges from 0.6 ~ 1468005 second.

As to manage connected user in the network, there is SA block and DA block supported to implement host block list. In addition, the SA block resource can change to SA secure as host permission list.

4.1 Layer 2 table

To support Layer 2 unicast, Layer 2 multicast, IPv4/IPv6 multicast forwarding, each packet type has been stored in L2 table using different format.

4.1.1 Entry format

The L2 entry format is shown in below:

Type	Hash Key		Table Content (width = 78bits)										
L2 Unicast	FID/VID (12)	MAC (48)	IP_MC (1)	IP6 (1)	MAC (48)	SLP(6)	Age(3)	SA Block SA Secure (1)	DA Block (1)	Static (1)	Suspend (1)	Nexthop (1)	VID (12)
L2 Multicast			IP_MC (1)	IP6 (1)	MAC (48)		IDX(12)						
IPv4/6 Multicast	SIP (32)	GIP(32)	IP_MC (1)	IP6 (1)	GIP (32)	SIP(20)	IDX(12)	VID (12)					
	FID/VID (12)	0+GIP (16+32)	IP_MC (1)	IP6 (1)	GIP (32)	Reserve (16)	IDX(12)						

Definition of L2 Unicast Fields

The entry is deemed a L2 Unicast entry if IP_MC = 0 and MAC[40] = 0.

Table 16: L2 Unicast Entry Fields

Field Name	Bits	Description
IP_MC	1	IP Multicast 0b0: Non IP multicast entry 0b1: IP multicast entry

IP6	1	IPv6 Packet 0b0: Non IPv6 entry 0b1: IPv6 entry
MAC	48	MAC address
SLP	6	Source Logical Port
AGE	3	Aging Time
SABLOCK/SASECURE	1	Source MAC address blocking/permission
DABLOCK	1	Destination MAC address blocking
STATIC	1	Static entry
SUSPEND	1	Suspending entry
NEXTHOP	1	Next Hop (refer to section L2 Next Hop Entry)
VID	12	For MAC-based N:1 VLAN aggregation(refer to section MAC-based N:1 VLAN Aggregation). The field is auto written by ASIC while learning a L2 unicast entry

Definition of L2 Multicast Fields

The entry is deemed a L2 Multicast entry if IP_MC = 0 and MAC[40] = 1.

Table 17: L2 Multicast Entry Fields

Field Name	Bits	Description
IP_MC	1	IP Multicast 0b0: Non IP multicast entry 0b1: IP multicast entry
IP6	1	IPv6 Packet 0b0: Non IPv6 entry 0b1: IPv6 entry
MAC	48	MAC address
IDX	12	Index to 4K PortMask table

Definition of IP Multicast Fields

The entry is deemed an IPv4 Multicast entry if IP_MC = 1 and IP6 = 0 while it is deemed an IPv6 Multicast entry if IP_MC = 1 and IP6 = 1.

Table 18: IP Multicast Entry Fields Correspond To SIP+GIP Lookup

Field Name	Bits	Description
IP_MC	1	IP Multicast 0b0: Non IP multicast entry 0b1: IP multicast entry
IP6	1	IPv6 Packet 0b0: Non IPv6 entry 0b1: IPv6 entry
GIP	32	Group IP address. For IPv6 multicast packet, system provides configuration to select 4 byte among 16 byte to be stored, please refer to section {SIP, GIP} Hash mode.
SIP	20	Source IP address (LSB 20 bits). For IPv6 multicast packet, system provides configuration to select 4 byte among 16 byte to be stored, please refer to section {SIP, GIP} Hash mode.
IDX	12	Index to 4K PortMask table
VID	12	VLAN ID

Table 19: IP Multicast Entry Fields Correspond To FID/VID+GIP

Field Name	Bits	Description
IP_MC	1	IP Multicast

		0b0: Non IP multicast entry 0b1: IP multicast entry
IP6	1	IPv6 Packet 0b0: Non IPv6 entry 0b1: IPv6 entry
GIP	32	Group IP address
RESERVED	16	Reserved
IDX	12	Index to 4K PortMask table

Layer 2 table can be accessed through following API.

API REFERENCE

```
rtk_l2_addr_add(uint32 unit, rtk_l2_ucastAddr_t *pL2_addr);
rtk_l2_addr_del(uint32 unit, rtk_vlan_t vid, rtk_mac_t *pMac);
rtk_l2_addr_get(uint32 unit, rtk_l2_ucastAddr_t *pL2_addr);
rtk_l2_addr_set(uint32 unit, rtk_l2_ucastAddr_t *pL2_addr);
rtk_l2_addr_delAll(uint32 unit, uint32 include_static);
rtk_l2_nextValidAddr_get(uint32 unit, int32 *pScan_idx, uint32 include_static, rtk_l2_ucastAddr_t
*pL2_data);

rtk_l2_mcastAddr_add(uint32 unit, rtk_l2_mcastAddr_t *pMcast_addr);
rtk_l2_mcastAddr_del(uint32 unit, rtk_vlan_t vid, rtk_mac_t *pMac);
rtk_l2_mcastAddr_get(uint32 unit, rtk_l2_mcastAddr_t *pMcast_addr);
rtk_l2_mcastAddr_set(uint32 unit, rtk_l2_mcastAddr_t *pMcast_addr)
rtk_l2_mcastAddr_add_with_index(uint32 unit, rtk_l2_mcastAddr_t *pMcast_addr);
rtk_l2_mcastAddr_get_with_index(uint32 unit, rtk_l2_mcastAddr_t *pMcast_addr);

rtk_l2_ipMcastAddr_add(uint32 unit, rtk_l2_ipMcastAddr_t *pIpmcast_addr);
rtk_l2_ipMcastAddr_del(uint32 unit, ipaddr_t sip, ipaddr_t dip, rtk_vlan_t vid);
rtk_l2_ipMcastAddr_get(uint32 unit, rtk_l2_ipMcastAddr_t *pIpmcast_addr);
rtk_l2_ipMcastAddr_set(uint32 unit, rtk_l2_ipMcastAddr_t *pIpmcast_addr);
rtk_l2_ipMcastAddr_add_with_index(uint32 unit, rtk_l2_ipMcastAddr_t *pIpMcast_addr);
rtk_l2_ipMcastAddr_get_with_index(uint32 unit, rtk_l2_ipMcastAddr_t *pIpMcast_addr);
rtk_l2_nextValidMcastAddr_get(uint32 unit, int32 *pScan_idx, rtk_l2_mcastAddr_t *pL2_data);
rtk_l2_nextValidIpMcastAddr_get(uint32 unit, int32 *pScan_idx, rtk_l2_ipMcastAddr_t *pL2_data);

rtk_l2_ip6McastAddr_add(uint32 unit, rtk_l2_ip6McastAddr_t *pIpmcast_addr);
rtk_l2_ip6McastAddr_del(uint32 unit, rtk_ipv6_addr_t sip, rtk_ipv6_addr_t dip, rtk_vlan_t vid);
rtk_l2_ip6McastAddr_get(uint32 unit, rtk_l2_ip6McastAddr_t *pIpmcast_addr);
rtk_l2_ip6McastAddr_set(uint32 unit, rtk_l2_ip6McastAddr_t *pIpmcast_addr);
rtk_l2_ip6McastAddr_add_with_index(uint32 unit, rtk_l2_ip6McastAddr_t *pIpMcast_addr);
rtk_l2_ip6McastAddr_get_with_index(uint32 unit, rtk_l2_ip6McastAddr_t *pIpMcast_addr);
rtk_l2_nextValidIp6McastAddr_get(uint32 unit, int32 *pScan_idx, rtk_l2_ip6McastAddr_t *pL2_data);
```

4.1.2 Unicast Entry Flush

The device provides control register L2_TBL_FLUSH_CTRL to flush L2 table entries, user can set FVID_CMP and PORT_CMP to clear or replace the matched entries. ENTRY_TYPE can be used to determine whether clear the software entry. All of them are regarded as additional conditions. If ACT is triggered without specifying FVID_CMP, PORT_CMP, all unicast entries in L2 table are flushed/replaced.

Table 20: L2_TBL_FLUSH_CTRL Register

Field Name	Bits	Description

STS	1	Start flush/replace action. It is cleared to 0 after flush/replace process is finished. 0b0: finish 0b1: start (busy)
ACT	1	The action to take. 0b0: flush 0b1: replace
FVID_CMP	1	Flush/Replace by matching FID/VID.
PORT_CMP	1	Flush/Replace by matching port.
ENTRY_TYPE	1	0b0: Static and Dynamic L2 unicast entry 0b1: Dynamic L2 unicast entry
FVID	12	FID/VID to be flushed or replaced.
PORT	6	Port ID to be flushed or replaced.
REPLACING_PORT	6	Port ID as the new port. The field is used only if ACT = 1.

API REFERENCE

```
rtk_l2_ucastAddr_flush(uint32 unit, rtk_l2_flushCfg_t *pConfig);
```

4.1.3 Aging

Each L2 unicast entry has the “Age” field. This field is updated when firstly learnt by device or SA lookup hits. When the value counts down to zero, the entry is aged out.

System provides global configuration L2_CTRL_1.AGE_UNIT to adjust the period. The adjustable range of aging out period is about 0.6s ~ 1,468,005s. The actual aging time is around L2_CTRL_1.AGE_UNIT *6 ~ L2_CTRL_1.AGE_UNIT *7.

Table 21: L2_CTRL_1.AGE_UNIT Register

Field Name	Bits	Description
AGE_UNIT	21	The granularity of AGE field in L2 table. AGE field decrease one every AGEUNIT* 0.1 seconds. Note: AGE_UNIT = 0 means disable aging.

The L2 table entries learnt from a port can be optionally aged-out by per-port setting: L2_PORT_AGING_OUT. AGING_OUT_EN.

Table 22: L2_PORT_AGING_OUT.AGING_OUT_EN Register

Field Name	Bits	Description
AGING_OUT_EN	1	Decide whether aging out those unicast entries associated to this port after a period of time. 0b0: disable 0b1: enable

In addition, there is a L2_CTRL_0.LINK_DOWN_P_INVLD register used to determine whether invalidate the corresponding L2 unicast entry when a port has link down.

Table 23: L2_CTRL_0.LINK_DOWN_P_INVLD Register

Field Name	Bits	Description
LINK_DOWN_P_INVLD	1	Invalidate the L2 unicast entries whose associated ports are link down 0b0: disable 0b1: enable

API REFERENCE

```
rtk_l2_aging_get(uint32 unit, uint32 *pAging_time);
rtk_l2_aging_set(uint32 unit, uint32 aging_time);
rtk_l2_portAgingEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable) ;
rtk_l2_portAgingEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable) ;
rtk_l2_flushLinkDownPortAddrEnable_get(uint32 unit, rtk_enable_t *pEnable);
rtk_l2_flushLinkDownPortAddrEnable_set(uint32 unit, rtk_enable_t enable);
```

4.1.4 Address Lookup and Learning

When a packet is received, its SA and DA are retrieved to lookup the L2 table. For SA lookup, if the SA is new to the device, a layer 2 entry with this SA and incoming port is learnt. For DA lookup, DA and some catachrestic are used to classify the packet type to be L2 unicast, L2 multicast, IPv4 multicast or IPv6 multicast. After packet type is classified, the corresponding hash key is used to find the matched Layer 2 entry to get the destination port information. If the SA/DA lookup process failed to find a matched entry, the lookup miss operation is taken.

Address Lookup

SA Lookup is the process verifying whether a unicast SA is existed. If a unicast SA exists, system gets information from this entry for decision later. If a unicast SA doesn't exist, Auto Learning process is executed. FID/VID and MAC are used as hash key for SA lookup.

DA lookup is the process searching L2 table and decides which port the packet should be forwarded. Since DA could be unicast, multicast (L2/ IPv4/ IPv6), or broadcast and the hash key of different packet types are different, the packet content is parsed to decide the packet type and the look up hash key.

The DA lookup process for different traffic type is as follows:

- **Broadcast**

1. DMAC = FF-FF-FF-FF-FF-FF
2. System floods this packet according the L2_FLD_PMSK.L2_BC_FLD_PMSK configuration.

- **IPv4 Multicast**

1. DMAC = 01-00-5E-XX-XX-XX
2. DMAC[23] = 0
3. Ethertype = 0x0800
4. If IP_MC_DIP_CHK = 1, check whether DIP belongs to Class-D scope (224~239.XX.XX.XX), and DIP[22:0] = DMAC[22:0].
5. This packet belongs to neither EAV Class A nor Class B traffic.(i.e., HSB.EAV_CLASS_A = 0 and HSB.EAV_CLASS_B = 0). Because EAV traffic is registered by mac address, hence should not lookup by IP address.
6. Hash key = {FID/VID, MAC} / {SIP, GIP} / {FID/VID, 0, GIP} defined by L2_CTRL_0. IPV4_MC_HASH_KEY_FMT
7. If a lookup hits, then forward this packet according to the portmask which IDX points to.
8. If a lookup not hit, take action specified by IP_MC_LM_ACT even the IPV4_MC_HASH_KEY_FMT is set to L2 Multicast mode.

- **IPv6 Multicast**

1. DMAC = 33-33-XX-XX-XX-XX
2. Ethertype = 0x86DD

3. This packet belongs to neither EAV Class A nor Class B traffic.
   (i.e., HSB.EAV_CLASS_A = 0 and HSB.EAV_CLASS_B = 0)
4. Hash key = {FID/VID, MAC} / {SIP, GIP} / {FID/VID, 0, GIP} defined by L2_CTRL_0. IPV6_MC_HASH_KEY_FMT.
5. If a lookup hits, then forward this packet according to the portmask which IDX points to.
6. If a lookup not hit, take action specified by IP6_MC_LM_ACT even the IPV6_MC_HASH_KEY_FMT is set to L2 Multicast mode.

For IGMP/MLD snooping and IPv6 forwarding, system supports three IPv4 and IPv6 Multicast modes configured by IPV4_MC_HASH_KEY_FMT and IPV6_MC_HASH_KEY_FMT.

1. {FID/VID, MAC} Hash mode

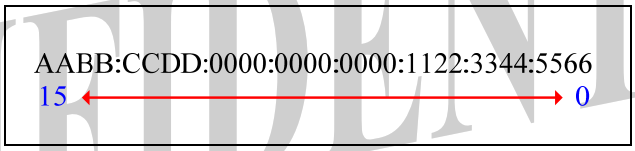
Treat IPv4/IPv6 Multicast as L2 Multicast. The fields are the same as L2 Multicast entry.

2. {SIP, GIP} Hash mode

- Use the SIP(32) and GIP(32) of an IPv4 Multicast packet as the Hash Key.
- For IPv6 multicast packet, 4 bytes of SIP and GIP are taken respectively in IPv6 Multicast packet as the Hash Key. The IPV6_MC_CARE_BYTE.SIP_CARE_BYTE and IPV6_MC_CARE_BYTE.DIP_CARE_BYTE are both four 4-bit nibbles which map to 4 of IPv6 IP's 16 byte. The device bases on these configurations to retrieve the corresponding 4 Bytes of SIP and DIP.

  By default, SIP Byte[5][2][1][0] and DIP Byte[3][2][1][0] are taken as the hash key. Thus the value of SIP_CARE_BYTE is 0x5210 while DIP_CARE_BYTE is 0x3210. Note that CARE_BYTE[0] maps to hash key[7:0], CARE_BYTE[1] maps to hash key[15:8], CARE_BYTE[2] maps to hash key[23:16] and CARE_BYTE[3] maps to hash key[31:24].

  Below example shows the endian of SIP/DIP byte, in the example byte[5][2][1][0] is 0x11445566.



3. {FID/VID, 0, GIP} Hash mode

Use the FID/VID and GIP(32) of an IPv4 Multicast packet as the Hash Key. For IPv6 multicast packet, the 4 bytes GIP retrieving is same as {SIP, GIP} Hash mode.

Table 24: IPV4/6_MC_HASH_KEY_FMT Register

Field Name	Bits	Description
IPV4_MC_HASH_KEY_FMT	2	IPv4 multicast hash key format. 0b00: same as L2 multicast (FID/VID + MAC) 0b01: SIP + GIP 0b10: FID/VID + 0 + GIP 0b11: reserved
IPV6_MC_HASH_KEY_FMT	2	IPv6 multicast hash key format. 0b00: same as L2 multicast (FID/VID + MAC) 0b01: SIP + GIP 0b10: FID/VID + 0 + GIP 0b11: reserved

Table 25: L2_IPV6_MC_IP_CARE_BYTE Register

Field Name	Bits	Description
DIP_CARE_BYTE	16	Cared bytes mask of DIP of IPv6 multicast packets that would be taken

		as DIP hash key. Four nibbles represent the positions of 4 bytes in DIP.
SIP_CARE_BYTE	16	Cared bytes mask of SIP of IPv6 multicast packets that would be taken as SIP hash key. Four nibbles represent the positions of 4 bytes in SIP.

API REFERENCE

```
rtk_l2_ipmcMode_get(uint32 unit, rtk_l2_ipmcMode_t *pMode);
rtk_l2_ipmcMode_set(uint32 unit, rtk_l2_ipmcMode_t mode);
rtk_l2_ip6mcMode_get(uint32 unit, rtk_l2_ipmcMode_t *pMode);
rtk_l2_ip6mcMode_set(uint32 unit, rtk_l2_ipmcMode_t mode);
rtk_l2_ip6CareByte_get(uint32 unit, rtk_l2_ip6_careByte_type_t type, uint32 *pCareByte);
rtk_l2_ip6CareByte_set(uint32 unit, rtk_l2_ip6_careByte_type_t type, uint32 careByte);
```

Address Learning

The packet is learnt follows per-port configuration L2_PORT_NEW_SALRN. NEW_SALRN and if the SA is new to the system, L2_PORT_NEW_SA_FWD. NEW_SA_FWD is taken.

When the NEW_SALRN = 0 (auto learn), only the first packet which is new to device is applied the NEW_SA_FWD configuration. For the packet with same SA receiving later, they are no longer new to device and the normal forwarding behavior is applied.

When the NEW_SALRN = 1 (learn as suspend), device writes the L2 entry as usual and set the Suspend bit to 1 additionally.

The suspending entry has the following characteristic:

- It could be aged out.
- It can not be replaced by other SMAC before it aged out.
- For the 1st packet, device takes action according to NEW_SA_FWD.
  For the 2nd, 3rd… packets, they are still regarded as new SMAC and the device updates the L2.AGE field but take different action according to different NEW_SA_FWD configuration.
  (i). NEW_SA_FWD = 0, => Forward
  (ii). NEW_SA_FWD = 1, => Drop
  (iii). NEW_SA_FWD = 2, => Drop
  (iv). NEW_SA_FWD = 3, => Forward

Table 26: L2_PORT_NEW_SALRN. NEW_SALRN Special SA Control Register

Field Name	Bits	Description
NEW_SALRN	1	L2 table learning behavior when receives a packet with new SMAC. 00b: ASIC Auto Learn 01b: learn as a suspend entry 10b: not Learn 11b: reserved

Table 27: L2_PORT_NEW_SALRN. NEW_SA_FWD Special SA Control Register

Field Name	Bits	Description
NEW_SA_FWD	2	Packet forwarding behavior when receives a packet with new SMAC. 0b00: forward 0b01: drop 0b10: trap 0b11: copy to CPU

If SA is Broadcast (FF-FF-FF-FF-FF-FF) or Multicast (MAC[40] = 1), SA Lookup/Learning is not performed and use the configuration L2_CTRL_0.MC_BC_SA_DROP and L2_CTRL_0.MC_BC_SA_TRAP to decide the forwarding behavior for the packet:

· If L2_CTRL_0.MC_BC_SA_DROP is 1, the BC/MC SA packet is dropped.
· If L2_CTRL_0.MC_BC_SA_DROP is 0 and L2_CTRL_0.MC_BC_SA_TRAP is 1, the packet is trapped.
· Otherwise, the packet is forwarded.

And if SA is all zero, system use the configuration L2_CTRL_0.ALL_ZERO_SA_DROP and L2_CTRL_0.ALL_ZERO_SA_TRAP to decide the forwarding behavior for the packet:

· If L2_CTRL_0.ALL_ZERO_SA_DROP is 1, the all zero SA packet is dropped.
· If L2_CTRL_0.ALL_ZERO_SA_DROP is 0 and L2_CTRL_0.ALL_ZERO_SA_TRAP is 1, the packet is trapped.
· Otherwise, the packet is forwarded.

Whether to perform SA Learning for all zero SA packets depends on the configuration of L2_CTRL_0.SA_ALL_ZERO_LRN.

Table 28: L2_CTRL_0 Special SA Control Register

Field Name	Bits	Description
MC_BC_SA_DROP	1	Decide whether drop the packet whose SA is multicast or broadcast. 0b0: disable (Not drop) 0b1: enable (Drop)
MC_BC_SA_TRAP	1	Decide whether trap the packet whose SA is all multicast or broadcast (only works when MC_BC_SA_DROP is disabled). 0b0: disable (Not trap) 0b1: enable (Trap)
ALL_ZERO_SA_DROP	1	Decide whether drop the packet whose SA is all zero. 0b0: disable (Not drop) 0b1: enable (Drop)
ALL_ZERO_SA_TRAP	1	Decide whether trap the packet whose SA is all zero (only works when ALL_ZERO_SA_DROP is disabled). 0b0: disable (Not trap) 0b1: enable (Trap)
SA_ALL_ZERO_LRN	1	The learning behavior for an all-zero mac-address (00-00-00-00-00-00). 0b0: not learn 0b1: learn

API REFERENCE

*rtk_l2_newMacOp_get(uint32 unit, rtk_port_t port, rtk_l2_newMacLrnMode_t *pLrnMode, rtk_action_t *pFwdAction);*
rtk_l2_newMacOp_set(uint32 unit, rtk_port_t port, rtk_l2_newMacLrnMode_t lrnMode, rtk_action_t fwdAction) ;
*rtk_l2_exceptionAddrAction_get(uint32 unit, rtk_l2_exceptionAddrType_t exceptType, rtk_action_t *pAction) ;*
rtk_l2_exceptionAddrAction_set(uint32 unit, rtk_l2_exceptionAddrType_t exceptType, rtk_action_t action) ;

4.1.5 Lookup Miss

For those lookup miss packets, device supports following action:

- Normal forward (i.e. Flooding).
- Drop
- Trap to CPU
- Copy to CPU

For L2 Multicast, IPv4 Multicast, IPv6 Muticast packets, system has separated configurations (through L2_PORT_LM_ACT.L2_UC_LM_ACT, L2_PORT_LM_ACT.L2_MC_LM_ACT, L2_PORT_LM_ACT.IP_MC_LM_ACT and L2_PORT_LM_ACT.IP6_MC_LM_ACT register).

If the configuration is set to forward, the forwarding (flooding) domain is defined by L2_UNKN_UC_FLD_PMSK, L2_UNKN_MC_FLD_PMSK, IP4_UNKN_MC_FLD_PMSK, or IP6_UNKN_MC_FLD_PMSK for unknown unicast, L2 unknown multicast, IPv4 unknown multicast, and IPv6 unknown multicast packets respectively. It should be noted that L2_UNKN_UC_FLD_PMSK is an L2 global register but L2_UNKN_MC_FLD_PMSK, IP4_UNKN_MC_FLD_PMSK, and IP6_UNKN_MC_FLD_PMSK are configurations in VLAN profile.

Table 29: L2_PORT_LM_ACT Register

Field Name	Bits	Description
L2_UC_LM_ACT	2	L2 unicast Packet lookup miss action Note: A L2 unicast packet is deemed to KNOWN if its DMAC and forwarding VID are equal to switch's MAC and the PVID of CPU port respectively. 0b00: forward (Flooding to L2_UNKN_UC_FLD_PMSK) 0b01: drop 0b10: trap 0b11: copy to CPU
L2_MC_LM_ACT	2	L2 Multicast Packet lookup miss action 0b00: forward (Flooding to L2_UNKN_MC_FLD_PMSK) 0b01: drop 0b10: trap 0b11: copy to CPU
IP_MC_LM_ACT	2	IPv4 Multicast Packet lookup miss action 0b00: forward (Flooding to IP4_UNKN_MC_FLD_PMSK) 0b01: drop 0b10: trap 0b11: copy to CPU
IP6_MC_LM_ACT	2	IPv6 Multicast Packet lookup miss action 0b00: forward (Flooding to IP6_UNKN_MC_FLD_PMSK) 0b01: drop 0b10: trap 0b11: copy to CPU

Table 30: Lookup Miss Forwarding PortMask Register

Field Name	Bits	Description
L2_UNKN_UC_FLD_PMSK	12	An index points to multicast table for retrieving unknown unicast flooding port mask.
L2_BC_FLD_PMSK	12	An index points to multicast table for retrieving broadcast flooding port mask.
L2_UNKN_MC_FLD_PMSK	12	An index points to multicast table for retrieving the unknown L2 multicast flooding port mask. The configuration is in VLAN Profile.
IP4_UNKN_MC_FLD_PMSK	12	An index points to multicast table for retrieving unknown IPv4 multicast flooding port mask. The configuration is in VLAN Profile.
IP6_UNKN_MC_FLD_PMSK	12	An index points to multicast table for retrieving unknown IPv6 multicast flooding port mask. The configuration is in VLAN Profile.

API REFERENCE

*rtk_l2_portLookupMissAction_get(uint32 unit, rtk_port_t port, rtk_l2_lookupMissType_t type, rtk_action_t *pAction);*
rtk_l2_portLookupMissAction_set(uint32 unit, rtk_port_t port, rtk_l2_lookupMissType_t type, rtk_action_t action) ;

*rtk_l2_lookupMissFloodPortMask_get(uint32 unit, rtk_l2_lookupMissType_t type, rtk_portmask_t * pFlood_portmask);*
rtk_l2_lookupMissFloodPortMask_add(uint32 unit, rtk_l2_lookupMissType_t type, rtk_port_t flood_port) ;
rtk_l2_lookupMissFloodPortMask_del(uint32 unit, rtk_l2_lookupMissType_t type, rtk_port_t flood_port) ;
*rtk_l2_lookupMissFloodPortMask_set_with_idx(uint32 unit, rtk_l2_lookupMissType_t type, uint32 idx, rtk_portmask_t *pFlood_portmask) ;*

4.2 Multicast Forwarding Table

The device supports a 4096 entry portmask table which is used for storing multicast portmask and also shared with ACL and VLAN profile. The table is utilized by all Layer 2 multicast and IP multicast entry, ACL multi-port redirect or VLAN profile.

Table 31: Multicast Forwarding PortMask Format

Field Name	Bits	Description
MC_PMSK	53	Multicast port mask

API REFERENCE

*rtk_l2_mcastFwdPortmask_get(uint32 unit, int32 index, rtk_portmask_t *pPortmask, uint32 *pCrossVlan) ;*
*rtk_l2_mcastFwdPortmask_set(uint32 unit, int32 index, rtk_portmask_t *pPortmask, uint32 crossVlan) ;*

4.3 Port Movement

4.3.1 Static Entry Port Movement

For static entry, there is a per-port configuration L2_PORT_STTC_MV_ACT that checks all static layer 2 entries with SLP on this port. If a static entry is SA lookup hit and the ingress port where the packet came from is different from the SLP, then the action specified by L2_PORT_STTC_MV_ACT (base on SLP port) is taken.

For example, assumes an L2 entry with Static = 1, SLP = 3 and L2_PORT_STTC_MV_ACT of port 3 is trap. If a packet matches this entry but came from port 5, then the configuration L2_PORT_STTC_MV_ACT of port 3 is taken to trap the packet.

Table 32: L2_PORT_STTC_MV_ACT Register

Field Name	Bits	Description
ACT	2	Static entry port move action. 0b00: forward 0b01: drop 0b10: trap 0b11: copy to CPU

API REFERENCE

*rtk_l2_staticPortMoveAction_get(uint32 unit, rtk_port_t port, rtk_action_t *pFwdAction) ;*
rtk_l2_staticPortMoveAction_set(uint32 unit, rtk_port_t port, rtk_action_t fwdAction) ;

4.3.2 Dynamic Entry Port Movement

For dynamic port move, device supports per-port configuration L2_PORT_MV_ACT. When dynamic port moving occurs, the configuration L2_PORT_MV_ACT of new port is taken to forward the packet. It should be noted that SLP and AGE fields are only updated when action is configured to forward or copy to CPU.

Table 33: L2_PORT_MV_ACT Register

Field Name	Bits	Description
ACT	2	Port move action. 0b00: forward 0b01: drop 0b10: trap 0b11: copy to CPU

And a special design for web authentication application is supplied, the per-port L2_PORT_MV_INVALIDATE.P_MV_INVLD is used to remove the L2 entry when receiving packets from ports other than original SLP, This configuration also follows the new port setting.

Table 34: L2_PORT_MV_INVALIDATE Register

Field Name	Bits	Description
P_MV_INVLD	2	Port move entry invalidate 0b00: disable 0b01: enable

API REFERENCE

```
rtk_l2_legalPortMoveAction_get(uint32 unit, rtk_port_t port, rtk_action_t *pFwdAction) ;
rtk_l2_legalPortMoveAction_set(uint32 unit, rtk_port_t port, rtk_action_t fwdAction);
rtk_l2_legalPortMoveFlushAddrEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable) ;
rtk_l2_legalPortMoveFlushAddrEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable) ;
```

4.3.3 Port Movement Forbidding

In some security applications, it may restrict a host which SA has learnt in L2 table from moving in or out a port. System provides per-port register L2_PORT_MV_FORBID. FORBID_EN. When the register is enabled on a port, a host cannot move in from other port or move out from this port. If a packet's ingress port is not equal to L2.SLP and one of the two ports' (ingress port or SLP) FORBID_EN is enabled, the L2_CTRL_0.FORBID_ACT is taken and the SA is not updated.

When the port movement actions conflict one another, the priority is:

L2_PORT_STATIC_MV_ACT > L2_CTRL_0.FORBID_ACT > L2_PORT_MV_ACT. Note that only when FORBID_ACT is not triggered, the module of dynamic port movement is performed and checked.

Table 35: L2_PORT_MV_FORBID Register

Field Name	Bits	Description
FORBID_EN	1	Enable/disable forbidding any port move in/out. 0b0: disable 0b1: enable

Table 36: L2_CTRL_0. FORBID_ACT Register

Field Name	Bits	Description
FORBID_ACT	2	Forbid action that takes effect only when

	L2_PORT_MV_FORBID.FORBID_EN is enabled and forbid event occurs. 0b00: forward 0b01: drop 0b10: trap 0b11: copy to CPU

API REFERENCE

```
rtk_l2_portDynamicPortMoveForbidEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);
rtk_l2_portDynamicPortMoveForbidEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
rtk_l2_dynamicPortMoveForbidAction_get(uint32 unit, rtk_action_t *pAction) ;
rtk_l2_dynamicPortMoveForbidAction_set(uint32 unit, rtk_action_t action) ;
```

4.4 Learning Constraint

The device provides system, per-port based and 32 VLAN-based learning constraint functions. When a port receives a packet with new SMAC, device checks all the learned numbers of the system, per-port, and VLAN concurrently. When any one of them reaches its own constraint number setting, the SMAC is not learnt and the corresponding constraint action is taken. If more than one constraint number setting overflows at the same time, the action priority is: VLAN > Port > System.

Here is an example,

(1) If a packet trigger port's and system's constraint action at the same time, and if Port's action is forward and System's action is drop, then the packet is forwarded.

(2) If a packet trigger VLAN's and system's constraint action at the same time, and if VLAN's action is drop and System's action is forward, then the packet is dropped.

The L2 unicast entries with following attributes don't be counted to learning counter:

- IP_MC = 1
- MAC[40] = 1
- STATIC = 1
- DABLOCK = 1
- SABLOCKSASECURE = 1
- SUSPEND = 1
- NEXTHOP = 1

It should be noted that the system/per-port/per-VLAN learning constraint functions only take effect when both L2_PORT_NEW_SALRN and L2_PORT_NEW_SA_FWD of the packet's ingress port are set to 0 (ASIC Auto learn, forward).

4.4.1 System Learning Constraint

System provides global register L2_LRN_CONSTRT with CONSTRT_NUM and LRN_CNT. If the LRN_CNT counter counts over the CONSTRT_NUM, the packet with new SA is not learnt and action according to L2_LRN_CONSTRT.ACT is taken.

Table 37: L2_LRN_CONSTRT Register

Field Name	Bits	Description
LRN_CNT	15	Learnt count. Read Only.

CONSTRT_NUM	15	Learning constraint number. 0x0: never learning 0x7FFF: unlimited
ACT	2	Learning constraint Action to take when LRN_CNT> CONSTRT_NUM. 0b00: forward 0b01: drop 0b10: trap 0b11: copy to CPU

API REFERENCE

```
rtk_l2_limitLearningCnt_get(uint32 unit, uint32 *pMac_cnt);
rtk_l2_limitLearningCnt_set(uint32 unit, uint32 mac_cnt);
rtk_l2_limitLearningCntAction_get(uint32 unit, rtk_l2_limitLearnCntAction_t *pAction) ;
rtk_l2_limitLearningCntAction_set(uint32 unit, rtk_l2_limitLearnCntAction_t action) ;
```

4.4.2 Per-Port Learning Constraint

System provides per-port CONSTRT_NUM and LRN_CNT register to limit and record L2 leant number. Whenever the device learns a SMAC from a port, it increases LRN_CNT counter of the port. If the counter counts over the CONSTRT_NUM, the new SMAC is not learnt and action according to L2_PORT_LRN_CONSTRT.ACT is taken.

If a dynamic entry is port moved, the learned counter of the original port is decreased and learned counter of the new port is increased. If user decreases the CONSTRT_NUM and causes the learnt counter LRN_CNT overflow, the new SMAC packet is not learnt but still be forwarded. And if a host is port moved to a LRN_CNT overflowed port, the L2 entry is not updated.

Table 38: L2_PORT_LRN_CONSTRT Register

Field Name	Bits	Description
LRN_CNT	15	Learnt count.
CONSTRT_NUM	15	Learning constraint number. 0x0: never learning 0x7FFF: unlimited
ACT	2	Learning constraint Action. 0b00: forward 0b01: drop 0b10: trap 0b11: copy to CPU

API REFERENCE

```
rtk_l2_portLimitLearningCnt_get(uint32 unit, rtk_port_t port, uint32 *pMac_cnt);
rtk_l2_portLimitLearningCnt_set(uint32 unit, rtk_port_t port, uint32 mac_cnt);
rtk_l2_portLimitLearningCntAction_get(uint32 unit, rtk_port_t port, rtk_l2_limitLearnCntAction_t *pAction) ;
rtk_l2_portLimitLearningCntAction_set(uint32 unit, rtk_port_t port, rtk_l2_limitLearnCntAction_t action) ;
```

4.4.3 VLAN Based Learning Constraint

In addition to learning constraint on port and system, the device provides 32 entries learning constraints on VLAN or VLAN-and-Port. The entry definition is as following:

Table 39:VLAN Based Learning Constraint Entry Format

Field Name	Bits	Description
FID/VID	12	The FID/VID to compare.
PORT	6	The Port ID to compare. 0x3f: Don't care
LRN_CNT	15	Learnt count.
CONSTRT_NUM	15	Learning constraint number. 0x0: never learning 0x7FFF: unlimited

And global supports a constraint action for these entries:

Table 40:VLAN Based Learning Constraint Action Register

Field Name	Bits	Description
ACT	2	Learning constraint on VLAN Action. System performs the action when LRN_CNT > CONSTRT_NUM. 0b00: forward 0b01: drop 0b10: trap 0b11: copy to CPU

Because L2 entries may be already learnt before the VLAN-based learning constraint is configured, user should disable L2 lerning of the device and then flush learnt L2 entries on the specific VLAN in order to make the LRN_CNT to be consistent with hareware L2 table.

API REFERENCE

```
rtk_l2_fidLimitLearningEntry_get(uint32 unit, uint32 fid_macLimit_idx, rtk_l2_fidMacLimitEntry_t
*pFidMacLimitEntry);
rtk_l2_fidLimitLearningEntry_set(uint32 unit, uint32 fid_macLimit_idx, rtk_l2_fidMacLimitEntry_t
*pFidMacLimitEntry);
rtk_l2_fidLearningCnt_get(uint32 unit, uint32 fid_macLimit_idx, uint32 *pNum) ;
rtk_l2_fidLearningCnt_reset(uint32 unit, uint32 fid_macLimit_idx) ;
rtk_l2_fidLearningCntAction_get(uint32 unit, rtk_l2_limitLearnCntAction_t *pAction);
rtk_l2_fidLearningCntAction_set(uint32 unit, rtk_l2_limitLearnCntAction_t action);
```

4.5 L2 Table Notification

Software may need to maintain a synchronous Layer 2 table as the device for quick response applications. The device provides L2 table notification mechanism to notify CPU once Layer 2 table is updated by hardware. Therefore, software can easily maintain a synchronous Layer 2 table without polling hardware by handling the notification events.

User can enable/disable the feature by a global configuration NOTIFICATION_EN. The following notification events are supported:

- aging out/ link-down flush/ port-move invalidate
- new learn
- port move
- suspend learn

Suspend entry learning event can be optionally enabled by register SUSPEND_NOTIFICATION_EN. And link down flush event can also be optionally enabled by register FLUSH_NOTIFY_EN.

The device internally has a 1K entries local buffer to store the notification events and the device DMA the events

to memory when bus is available. If the device's local buffer runs out, a LocalBuf-run-out interrupt is triggered.
To minimize the chance of LocalBuf-run-out, system supports a "backpressure" mechanism. If accumulated notification event entries beyond the BP_THR, then backpressure is started. Once backpressure mechanism is started, L2 table stops aging-out entries (but can still learn) until the number of notification event entries is lower than BP_THR.

The system provides friendly API that user can take use of the notification by enabled the notification and register event handlers without caring the detail such as allocating memory for NBuf or handling the OWN bit. Programming Example 4.7.11 demonstrates how to use L2 notification API.

Table 41: L2 Notification Related Registers

Field Name	Bits	Description
SUSPEND_NOTIFICATION_EN	1	Enable/Disable susupend entry new learn event notification. 0: disable 1: enable
BP_THR	10	The threshold to start backpressure. 0x3ff: disable backpressure
FIFO_EMPTY	1	The internal FIFO is empty. 0: empty 1: not empty
NOTIFICATION_EN	1	Enable/Disable Notification. 0: disable 1: enable
FLUSH_NOTIFY_EN	1	Notify CPU about link down flush events. 0: disable 1: enable

API REFERENCE

```
rtk_l2_notificationEnable_get(uint32 unit, rtk_enable_t *pEnable);
rtk_l2_notificationEnable_set(uint32 unit, rtk_enable_t enable);
rtk_l2_notificationTypeEnable_get(uint32 unit, rtk_l2_notifyType_t type, rtk_enable_t *pEnable) ;
rtk_l2_notificationTypeEnable_set(uint32 unit, rtk_l2_notifyType_t type, rtk_enable_t enable) ;
rtk_l2_notificationBackPressureThresh_get(uint32 unit, uint32 *pThresh);
rtk_l2_notificationBackPressureThresh_set(uint32 unit, uint32 thresh);
rtk_l2_notificationFifoEmptyStatus_get(uint32 unit, rtk_l2_notifyFifoStatus_t *pStatus);
rtk_l2_notificationEventHandler_register(uint32 unit, rtk_l2_notification_callback_t notification_callback,
rtk_l2_notification_bufferRunOut_callback_t bufferRunOut_callback);
rtk_l2_notificationEventHandler_unregister(uint32 unit);
```

4.6 Blocking

For L2 unicast entry format, there are SA Block/SA Secure bit and DA block bit for SA Block/SA Secure and DA Block function. SA Block and SA Secure share the same bit field of the table entry and are switched by global configuration register L2_CTRL_0.SECURE_SA.

4.6.1 SA Block

When L2_CTRL_0.SECURE_SA is disabled and the SA Block bit of L2 unicast entry is TRUE, the packet whose source MAC is the same as the table entry is dropped. SA Block entry is age-able and it stays valid after age-out. Unlike normal dynamic L2 unicast entry, the SLP is set to 0x3F while age-out. Initializes SA Block entries with SLP = 0x3F allows a DA lookup hit packet to flood so that other hosts can still forward packet to the SA blocked hosts. Note:

SA Block field is 0 for a hardware auto learnt entry.
SA_BLK_EN is a per port register that determines if a hit SA block entry should take effect. For example, if a L2 entry with MAC=00:11:22:33:44:55 and SA block = 1 is hit for a packet receiving from port 1. If SA_BLK_EN of port 1 is enabled, the packet would be dropped and if SA_BLK_EN of port 1 is disabled, the packet would be forwarded.

Table 42: SA Block Enable Register

Field Name	Bits	Description
SA_BLK_EN	1	Per port enable/disable SA block function. 0b0: disable 0b1: enable

API REFERENCE

```
rtk_l2_secureMacMode_get(uint32 unit, rtk_enable_t *pEnable);
rtk_l2_secureMacMode_set(uint32 unit, rtk_enable_t enable);
rtk_l2_portMacFilterEnable_get(uint32 unit, rtk_port_t port, rtk_l2_macFilterMode_t filterMode, rtk_enable_t *pEnable);
rtk_l2_portMacFilterEnable_set (uint32 unit, rtk_port_t port, rtk_l2_macFilterMode_t filterMode, rtk_enable_t enable) ;
```

4.6.2 SA Secure

In SA Secure mode (SA white list), only the secure host traffic is permitted to forward to its destination, traffic sent by non-secure host should be dropped. This can be achieved by configuring L2_PORT_NEW_SA_FWD.NEW_SA_FWD to drop the SA lookup miss packet. Besides, L2_CTRL_0.SECURE_SA should be enabled and corresponding L2 entries (SA_Secure=1) for those secure hosts should be inserted for SA lookup hit.
Note: SA Secure field is 0 for a hardware auto learnt entry.

Table 43: L2_CTRL_0.SECURE_SA Register

Field Name	Bits	Description
SECURE_SA	1	Enable/Disable Secure SA mode. 0b0: disable 0b1: enable

API REFERENCE

```
rtk_l2_secureMacMode_get(uint32 unit, rtk_enable_t *pEnable);
rtk_l2_secureMacMode_set(uint32 unit, rtk_enable_t enable);
```

4.6.3 DA Block

When DA Block of a L2 unicast entry is TRUE, a packet whose destination MAC lookup hits the entry is dropped (But SMAC is still learnt because the SA Learning is earlier than DA Lookup).
DA_BLK_EN is a per port register that determines if a hit DA block entry should take effect. DA_BLK_EN is referred by packet's egress port. For example, if DA_BLK_EN of port 1, 2, 3 are enabled, for port 4, 5, 6 are disabled, and the SLP is 0x3F (means all ports). The DA block hit packet coming from port 1 would flood to port 4, 5, 6, and the DA block hit packet coming from port 4 would flood to port 5, 6.

Table 44: SA Block Enable Register

Field Name	Bits	Description

DA_BLK_EN	1	Per port enable/ disable DA block function. 0b0: disable 0b1: enable

API REFERENCE

```
rtk_l2_portMacFilterEnable_get(uint32 unit, rtk_port_t port, rtk_l2_macFilterMode_t filterMode, rtk_enable_t *pEnable);
rtk_l2_portMacFilterEnable_set (uint32 unit, rtk_port_t port, rtk_l2_macFilterMode_t filterMode, rtk_enable_t enable) ;
```

4.7 Application Example

Here are some applications mostly used in a layer 2 switch for quick reference.

4.7.1 Add/Delete Static entry

To add a static entry with MAC=00:11:22:33:44:55, VID=1

```
/* variable declaration */
uint32 unit = 0;
rtk_l2_ucastAddr_t l2_uAddr;
uint32 sa_block = FALSE;
uint32 da_block = FALSE;
uint32 is_static = TRUE;
uint32 nexthop = FALSE;
uint32 suspend = FALSE;

/* unicast entry configuration */
memcpy(l2_uAddr.mac.octet, 0x001122334455, 6);
l2_uAddr.vid = 1;
l2_uAddr.port = 0;
l2_uAddr.agg_vid = 0 ;
If(sa_block)
    l2_uAddr.flags |= RTK_L2_UCAST_FLAG_SA_BLOCK;
If(da_block)
    l2_uAddr.flags |= RTK_L2_UCAST_FLAG_DA_BLOCK;
If(is_static)
    l2_uAddr.flags |= RTK_L2_UCAST_FLAG_STATIC;
If(nexthop)
    l2_uAddr.flags |= RTK_L2_UCAST_FLAG_NEXTHOP;
If(suspend)
    l2_uAddr.state |= RTK_L2_UCAST_STATE_SUSPEND;

/* add unicast entry */
rtk_l2_addr_add(unit, &l2_uAddr);
```

To delete a static entry with MAC=00:11:22:33:44:55, VID=1

```
/* variable declaration */
uint32 unit = 0;
rtk_mac_t mac;

/* unicast entry configuration */
```

```
memcpy(mac.octet, 0x001122334455, 6);

/* delete unicast entry */
rtk_l2_addr_del(unit, 1, & mac);
```

4.7.2 Add/Delete L2 Multicast entry

To add a L2 multicast entry with MAC=01:00:22:33:44:55, VID=1, forward portmask=port 1,3,5:

```
/* variable declaration */
rtk_l2_mcastAddr_t    mcast_data;

/* multicast entry configuration */
memcpy(mcast_data.mac.octet, 0x010022334455, 6);
mcast_data.rvid = 1;
mcast_data.portmask.bits[0] = (1 << 0) | (1 << 2) | (1 << 4);

/* add multicast entry */
rtk_l2_mcastAddr_add(unit, &mcast_data);
```

To delete a L2 multicast entry with MAC=01:00:22:33:44:55, VID=1:

```
/* variable declaration */
uint32 unit = 0;
rtk_mac_t mac;

/* multicast entry configuration */
memcpy(mac.octet, 0x010022334455, 6);

/* delete multicast entry */
rtk_l2_mcastAddr_del (unit, 1, & mac);
```

4.7.3 Add/Delete IPv4 Multicast entry

To add an IPv4 multicast entry with MAC+VID for MAC=01:00:22:33:44:55, VID=1, forward portmask=port 1,3,5:

```
/* variable declaration */
uint32 unit = 0;
rtk_l2_mcastAddr_t mcast_data;

/* multicast entry configuration */
memcpy(mcast_data.mac.octet, 0x010022334455, 6);
mcast_data.rvid = 1;
mcast_data.portmask.bits[0] = (1 << 0) | (1 << 2) | (1 << 4);

/* set ip multicast hash mode to MAC+VID */
rtk_l2_ipmcMode_set(unit, LOOKUP_ON_FVID_AND_MAC)

/* add multicast entry */
rtk_l2_mcastAddr_add(unit, &mcast_data);
```

To add an IPv4 multicast entry with GIP+SIP for SIP=140.0.0.1,DIP=239.1.1.1, forward portmask=port 2,4,6:

```
/* variable declaration */
uint32 unit = 0;
rtk_l2_ipMcastAddr_t ip_mcast_data;

/* multicast entry configuration */
ip_mcast_data.sip = (140 << 24) |( 0 << 16) | (0 << 8) | 1;
ip_mcast_data.dip = (239 << 24) |( 1 << 16) | (1 << 8) | 1;
ip_mcast_data.portmask.bits[0] = (1 << 1) | (1 << 3) | (1 << 5);

/* set ip multicast hash mode to GIP+SIP */
rtk_l2_ipmcMode_set(unit, LOOKUP_ON_DIP_AND_SIP)

/* add multicast entry */
rtk_l2_ipMcastAddr_add(unit, &ip_mcast_data);
```

To add an IPv4 multicast entry with VID+GIP for VID=100, DIP=239.1.1.1, forward portmask=port 2,4,6:

```
/* variable declaration */
uint32 unit = 0;
rtk_l2_ipMcastAddr_t ip_mcast_data;

/* multicast entry configuration */
ip_mcast_data.rvid = 100;
ip_mcast_data.dip = (239 << 24) |( 1 << 16) | (1 << 8) | 1;
ip_mcast_data.portmask.bits[0] = (1 << 1) | (1 << 3) | (1 << 5);

/* set ip multicast hash mode to GIP+SIP */
rtk_l2_ipmcMode_set(unit, LOOKUP_ON_DIP_AND_FVID)

/* add multicast entry */
rtk_l2_ipMcastAddr_add(unit, &ip_mcast_data);
```

To delete an IPv4 multicast entry with GIP+SIP for SIP=140.0.0.1,DIP=239.1.1.1:

```
/* variable declaration */
uint32 unit = 0;
uint32 sip;
uint32 dip;

/* multicast entry configuration */
sip = (140 << 24) |( 0 << 16) | (0 << 8) | 1;
dip = (239 << 24) |( 1 << 16) | (1 << 8) | 1;

/* set ip multicast hash mode to GIP+SIP */
rtk_l2_ipmcMode_set(unit, LOOKUP_ON_DIP_AND_SIP)

/* delete multicast entry */
rtk_l2_ipMcastAddr_del(unit, sip, dip, 0);
```

4.7.4 Add/Delete IPv6 Multicast entry

To add an IPv6 multicast entry with MAC+VID for MAC=01:00:22:33:44:55, VID=1, forward portmask=port 1,3,5:

```
/* variable declaration */
uint32 unit = 0;
rtk_l2_mcastAddr_t mcast_data;

/* multicast entry configuration */
memcpy(mcast_data.mac.octet, 0x010022334455, 6);
mcast_data.rvid = 1;
mcast_data.portmask.bits[0] = (1 << 0) | (1 << 2) | (1 << 4);

/* set ip multicast hash mode to MAC+VID */
rtk_l2_ip6mcMode_set( unit, LOOKUP_ON_FVID_AND_MAC)

/* add multicast entry */
rtk_l2_mcastAddr_add(unit, &mcast_data);
```

To add an IPv6 multicast entry with GIP+SIP for SIP=2001::73, DIP= ff09::5, forward portmask=port 2,4,6:

```
/* variable declaration */
uint32 unit = 0;
rtk_l2_ip6McastAddr_t ip6_mcast_data;

/* multicast entry configuration */
ip_mcast_data.sip.ipv6_addr[0] = 0x20;
ip_mcast_data.sip.ipv6_addr[1] = 0x01;
ip_mcast_data.sip.ipv6_addr[15] = 0x73;
ip_mcast_data.dip.ipv6_addr[0] = 0xff;
ip_mcast_data.dip.ipv6_addr[1] = 0x09;
ip_mcast_data.dip.ipv6_addr[15] = 0x5;
ip_mcast_data.portmask.bits[0] = (1 << 1) | (1 << 3) | (1 << 5);

/* set ip multicast hash mode to GIP+SIP */
rtk_l2_ip6mcMode_set (unit, LOOKUP_ON_DIP_AND_SIP)

/* add multicast entry */
rtk_l2_ip6McastAddr_add(unit, &ip6_mcast_data);
```

To add an IPv6 multicast entry with VID+GIP for VID=100, DIP=ff09::5, forward portmask=port 2,4,6:

```
/* variable declaration */
uint32 unit = 0;
rtk_l2_ip6McastAddr_t ip6_mcast_data;

/* multicast entry configuration */
ip_mcast_data.rvid = 100;
ip_mcast_data.dip.ipv6_addr[0] = 0xff;
ip_mcast_data.dip.ipv6_addr[1] = 0x09;
ip_mcast_data.dip.ipv6_addr[15] = 0x5;
```

```
ip_mcast_data.portmask.bits[0] = (1 << 1) | (1 << 3) | (1 << 5);

/* set ip multicast hash mode to GIP+SIP */
rtk_l2_ip6mcMode_set (unit, LOOKUP_ON_DIP_AND_FVID)

/* add multicast entry */
rtk_l2_ip6McastAddr_add(unit, &ip_mcast_data);
```

To delete an IPv6 multicast entry with GIP+SIP for SIP=2001::73, DIP= ff09::5:

```
/* variable declaration */
uint32 unit = 0;
rtk_ipv6_addr_t sip;
rtk_ipv6_addr_t dip;

/* multicast entry configuration */
sip.ipv6_addr[0] = 0x20;
sip.ipv6_addr[1] = 0x01;
sip.ipv6_addr[15] = 0x73;
dip.ipv6_addr[0] = 0xff;
dip.ipv6_addr[1] = 0x09;
dip.ipv6_addr[15] = 0x5;

/* set ip multicast hash mode to GIP+SIP */
rtk_l2_ip6mcMode_set(uint32 unit, LOOKUP_ON_DIP_AND_SIP)

/* delete multicast entry */
rtk_l2_ip6McastAddr_del(unit, sip, dip, 0);
```

4.7.5 Web Authentication

```
/* variable declaration */
uint32 unit = 0;
rtk_port_t port ;
rtk_l2_newMacLrnMode_t lrnMode ;
rtk_action_t fwdAction;

/* variable configuration */
lrnMode = NOT_LEARNING;
fwdAction = ACTION_TRAP2CPU;

/* disable L2 learning and trap new SA/ port move packet, enable port move flush */
FOR_EACH_LOGIC_PORT(port)
{
    rtk_l2_newMacOp_set(unit, port, lrnMode, fwdAction);
    rtk_l2_legalPortMoveAction_set(unit, port, fwdAction);
    rtk_l2_legalPortMoveFlushAddrEnable_set(unit, port, ENABLED) ;
}

/* software add authorized L2 entry */
rtk_l2_addr_add(unit, &l2_uAddr);
```

4.7.6 Port Base mac constraint

Limit port 10 to learn at most 50 dynamic entries, and drop the further new SA packets:

```
/* variable declaration */
uint32 unit = 0;
rtk_port_t port = 9;
uint32 mac_cnt = 50;
rtk_l2_limitLearnCntAction_t action = LIMIT_LEARN_CNT_ACTION_DROP ;

/* add multicast entry */
rtk_l2_portLimitLearningCnt_set(unit, port, mac_cnt);
rtk_l2_portLimitLearningCntAction_set( unit, port, action);
```

4.7.7 VLAN base mac constraint

Limit VLAN 100 to learn at most 50 dynamic entries, and trap the further new SA packets:

```
/* variable declaration */
uint32 unit = 0;
uint32 fid_macLimit_idx = 0;
rtk_l2_fidMacLimitEntry_t fidMacLimitEntry ;
rtk_l2_limitLearnCntAction_t action;

/* variable configuration */
fidMacLimitEntry.fid = 100;
fidMacLimitEntry.maxNum = 50
action = ACTION_TRAP2CPU;

/* set VLAN based mac constraint entry and action */
rtk_l2_fidLimitLearningEntry_set(unit, fid_macLimit_idx, &fidMacLimitEntry);
rtk_l2_fidLearningCntAction_set(unit, action);
```

4.7.8 SA Block

Enable SA block on all ports and add an SA block entry:

```
/* variable declaration */
uint32 unit = 0;
rtk_port_t port;

/* unicast entry configuration with SA block*/
memcpy(l2_uAddr.mac.octet, 0x001122334455, 6);
l2_uAddr.vid = 1;
l2_uAddr.port = 0x3f;
l2_uAddr.agg_vid = 0 ;
l2_uAddr.flags |= TRUE << RTK_L2_UCAST_FLAG_SA_BLOCK;

/* enable SA block for all ports */
FOR_EACH_LOGIC_PORT(port)
{
      rtk_l2_portMacFilterEnable_set (unit, port, MAC_FILTER_MODE_SA, ENABLED) ;
}

/* add unicast entry */
rtk_l2_addr_add(unit, &l2_uAddr);
```

4.7.9 DA Block

Enable DA block on all ports and add an DA block entry:

```
/* variable declaration */
uint32 unit = 0;
rtk_port_t port;

/* unicast entry configuration with DA block*/
memcpy(l2_uAddr.mac.octet, 0x001122334455, 6);
l2_uAddr.vid = 1;
l2_uAddr.port = 0x3f;
l2_uAddr.agg_vid = 0 ;
l2_uAddr.flags |= TRUE << RTK_L2_UCAST_FLAG_DA_BLOCK;

/* enable DA block for all ports */
FOR_EACH_LOGIC_PORT(port)
{
    rtk_l2_portMacFilterEnable_set (unit, port, MAC_FILTER_MODE_DA, ENABLED) ;
}

/* add unicast entry */
rtk_l2_addr_add(unit, &l2_uAddr);
```

4.7.10 SA Secure

Enable SA secure and add an SA secure entry:

```
/* variable declaration */
uint32 unit = 0;
rtk_port_t port;

/* unicast entry configuration with SA secure*/
memcpy(l2_uAddr.mac.octet, 0x001122334455, 6);
l2_uAddr.vid = 1;
l2_uAddr.port = 0x3f;
l2_uAddr.agg_vid = 0 ;
l2_uAddr.flags |= TRUE << RTK_L2_UCAST_FLAG_SA_BLOCK; /*SA secure and SA block share the
same bit and the definition of the bit is decided by SA secure enable status*/

/* enable SA secure */
rtk_l2_secureMacMode_set(unit, ENABLED);

/* add unicast entry */
rtk_l2_addr_add(unit, &l2_uAddr);
```

4.7.11 L2 Notification

Enable L2 notification, suspend learn notification, and register notification callback function and buffer-run-out callback function:

```
/* notify callback function */
int32 l2_notifyHandler(uint32 unit, rtk_l2_notifyEventCollectArray_t *pEventCollectArray)
{
    int i;
    rtk_l2_notifyEvent_t *pEvent;

    for (i = 0; i < pEventCollectArray->entryNum; i++)
    {
        pEvent = &pEventCollectArray->eventArray[i];
        printk("type:%d vid:%d mac:%x-%x-%x slp:%d\n", pEvent->type, pEvent->fidVid,
            pEvent->mac.octet[3], pEvent->mac.octet[4], pEvent->mac.octet[5], pEvent->slp);
    }

    return SUCCESS ;
}

/* buffer run out callback function */
int32 l2_notifyBufRunOutHandler(uint32 unit, rtk_l2_notifyBufferType_t type)
{
    printk("type:%d buffer run out\n", type);
    return SUCCESS;
}

/* variable declaration */
uint32 unit = 0;
rtk_port_t port;

rtk_l2_notificationEnable_set(unit, ENABLED);
rtk_l2_notificationTypeEnable_set( unit, L2_NOTIFY_TYPE_SUSPEND_LEARN, ENABLED) ;
rtk_l2_notificationEventHandler_register(unit, l2_notifyHandler, l2_notifyBufRunOutHandler);
```

5 Access Control List

The device supports an ACL engine to process and classify packets at wire speed. ACL is used for operating and managing traffic for flow-based applications such as flow-based access control list (ACL), flow-based Quality of Service (QoS), flow-based rate limiting, flow-based mirroring, and so on.

ACL engine is mainly composed of compare engine and action engine. The ACL compare engine filters packet data down to bit level and the action engine executes the command for the qualified packet for specific applications.

5.1 ACL Overview

The device supports 2304 (2K+256) entries in total which are divided into 18 ACL blocks, therefore, each block contains 128 entries. Each block provides an independent lookup and power enable configurations.

Table 45: ACL_BLK_LOOKUP_CTRL Register

Field Name	Bits	Description
LUT_EN	1	Enable lookup for a specific ACL block. 1'b0: disable 1'b1: enable

Table 46: PS_ACL_PWR_CTRL Register

Field Name	Bits	Description
ACL_PWR_EN	1	Enable power for a specific ACL block. 1'b0: disable 1'b1: enable

The device provides 5 set of pre-defined and 3 set of user-defined (configurable) templates. The template is the compare key to filter packet. Each block can be matched for 2 set of templates and each entry has a fixed field to further select one of these two templates to map.

Table 47: ACL_BLK_TMPLTE_CTRL Register

Field Name	Bits	Description
BLK_TMPLATE1	3	Maps block to the first template.
BLK_TMPLATE2	3	Maps block to the second template.

API REFERENCE

```
rtk_acl_blockLookupEnable_get(uint32 unit, uint32 block_idx, rtk_enable_t *pEnable)
rtk_acl_blockLookupEnable_set(uint32 unit, uint32 block_idx, rtk_enable_t enable)

rtk_acl_blockPwrEnable_get(uint32 unit, uint32 block_idx, rtk_enable_t *pEnable)
rtk_acl_blockPwrEnable_set(uint32 unit, uint32 block_idx, rtk_enable_t enable)

rtk_acl_templateSelector_get(uint32 unit, uint32 block_idx, rtk_acl_templateIdx_t *pTemplate_idx)
rtk_acl_templateSelector_set(uint32 unit, uint32 block_idx, rtk_acl_templateIdx_t template_idx)
```

5.2 ACL Partition

The device supports both Ingress and Egress ACL. All the blocks are divided into Ingress or Egress ACL by

CUTELINE register. Block number less than CUTELINE is considered Ingress ACL blocks, otherwise is considered Egress ACL blocks. Below figure shows the position where the Ingress and Egress ACL modules are located.

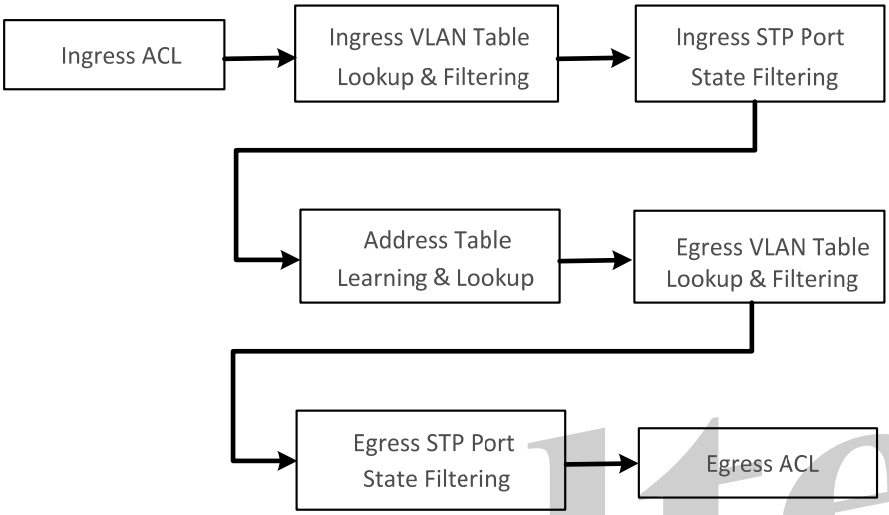


Figure 15: Ingress and Egress ACL Module Location

API REFERENCE

```
rtk_acl_partition_get(uint32 unit, uint32 *pPartition)
rtk_acl_partition_set(uint32 unit, uint32 partition)
```

5.3 ACL Functional Block

The receiving packet is first processed by Compare Engine and outputs hit results. The results are then further processed by ACL entry operations and block operations. Once the final hit results are given, the Action Engine executes the actions for the hits entries after running action arbitration.

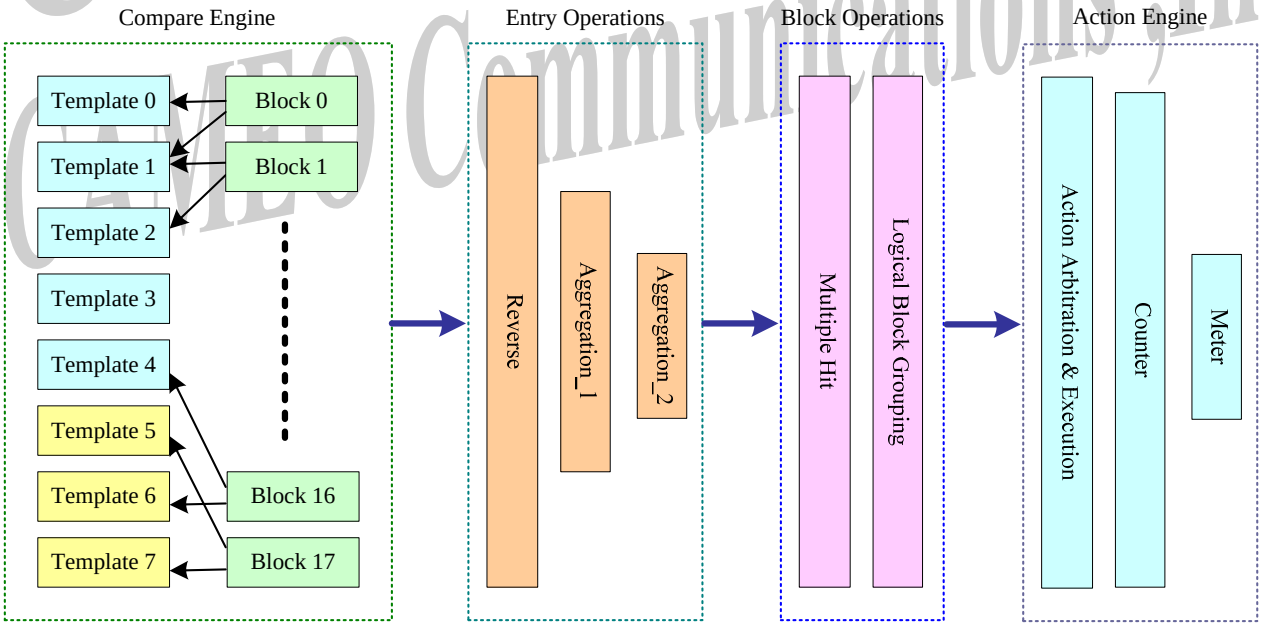


Figure 16: ACL Functional Block

5.4 ACL Template

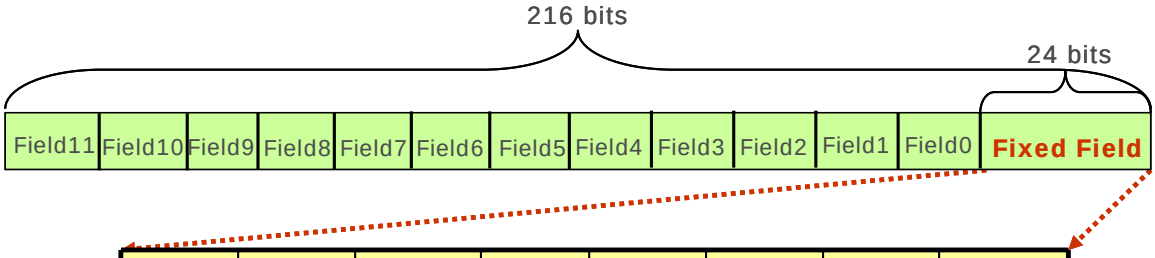
An ACL entry is composed of Rule Data, Operations and Actions. The length of each entry's rule data (compare key) is 216–bits (192-bits template and 24-bits fixed field).

The 192-bits width template is composed of 12 16-bits width template fields. For user-defined templates, each field can be configured one template field type to filter packet characteristics. There are three kinds of field types: fixed, shared, and egress. We will take a look at these field types in following sections.

5.4.1 Fixed Field Type

The 192-bits rule data of the ACL entry is defined according to the mapped template. Each ACL entry also contains a 24-bits width fixed field which is independent of templates. It is used to qualify following characteristics:

- Template ID: select the mapping template
- Inner Tag or Inner Priority Tag Packet
- Outer Tag or Outer Priority Tag Packet
- Inner Untag or Inner Priority Tag Packet
- Outer Untag or Outer Priority Tag Packet
- Frame Type: ARP (exclude RARP), IPv4, IPv6, Other
- L2 Frame Type: Ethernet, LLC_SNAP or LLC_OTHER
- L4 Frame Type: UDP, TCP, ICMP/ICMPv6, IGMP or L4 unknown
- IP Non-Zero Offset: IPv4 or IPv6 packet offset value is non-zero.
- Switch MAC: DMAC of a packet is destined to one of the switch MAC address.
- Management VLAN: Packet's forwarding VID matches CPU port's P-VID. The forwarding VID is either from tagged packet or P-VID of untagged packet.
- SPN: Source port number
- SPMM01: The results of source port mask entry 0 and 1.



Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Outer Untag or Outer Pri-Tag	Inner Untag or Inner Pri-Tag	Outer Tag or Outer Pri-Tag	Inner Tag or Inner Pri-Tag	00: Ethernet 01: LLC_SNAP 10: LLC_Other 11: Reserved		**Template ID** Bit 1 is reserved	
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
MGNT VLAN	**Switch MAC**	IP4/IP6 offset!=0	000: UDP 001: TCP 010: ICMP/ICMPv6 011: IGMP 1XX: L4 unknown			00: ARP 01: L2 ONLY 10: IPv4 11: IPv6	
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Result of Source Portmask Entry 0,1		Source Logical Port					

Figure 17: ACL Entry Rule Data

API REFERENCE

Corresponding Field Name of Fixed Field Type:
USER_FIELD_TEMPLATE_ID
USER_FIELD_ITAG_EXIST
USER_FIELD_OTAG_EXIST
USER_FIELD_ITAG_FMT
USER_FIELD_OTAG_FMT
USER_FIELD_FRAME_TYPE
USER_FIELD_FRAME_TYPE_L2
USER_FIELD_L4_PROTO
USER_FIELD_IP_NONZEROOFFSET
USER_FIELD_SWITCHMAC
USER_FIELD_MGNT_VLAN
USER_FIELD_SPN
USER_FIELD_SPMM_0_1

*rtk_acl_ruleEntryField_read(uint32 unit, rtk_acl_phase_t phase, rtk_acl_id_t entry_idx, rtk_acl_fieldType_t type, uint8 *pData, uint8 *pMask)*
*rtk_acl_ruleEntryField_write(uint32 unit, rtk_acl_phase_t phase, rtk_acl_id_t entry_idx, rtk_acl_fieldType_t type, uint8 *pData, uint8 *pMask)*

```
rtk_acl_ruleEntryField_get(
    uint32                  unit,
    rtk_acl_phase_t         phase,
    rtk_acl_id_t            entry_idx,
    uint8                   *pEntry_buffer,
    rtk_acl_fieldType_t     type,
    uint8                   *pData,
```

```
    uint8                   *pMask)
rtk_acl_ruleEntryField_set(
    uint32                  unit,
    rtk_acl_phase_t         phase,
    rtk_acl_id_t            entry_idx,
    uint8                   *pEntry_buffer,
    rtk_acl_fieldType_t     type,
    uint8                   *pData,
    uint8                   *pMask)
```

5.4.2 Share Field Type

The share field types shown in below table are for both Ingress and Egress ACL. The table is organized by four columns:
Template Field: describes the template field name.
Field Type: describes the field type name.
Description: describes the meaning of the field.
Template Field Position Limitation: defines the specific index (0~11) that the template field can be located for user-defined templates. If it is empty, the template field can appear in any index.

The table also shows the relationship between template field and field type. One filed type may consume several template fields while several field types may be located in the same template field.

Table 48: Share Field Type and Template Field Mapping Table

Field Type (USER_FIELD_XXX)	Template Field (TMPLTE_FIELD_XXX)	Description	Template Field Position Limitation
SPMM	SPMMASK	Comparison result of source port bitmap mask configurations	
SPM	SPM0	Source Port Mask [15:0]	0/4/8
	SPM1	Source Port Mask [31:16]	1/5/9
	SPM2	Source Port Mask [47:32]	2/6/10
	SPM3	Source Port Mask [52:48]	3/7/11
ETAG_EXIST	SPM3	Extra-Tagged	3/7/11
PORT_RANGE		TCP/UDP Port Range Check Mask	
DMAC	DMAC0	DMAC[15:0]	0/3/6/9
	DMAC1	DMAC[31:16]	1/4/7/10
	DMAC2	DMAC[47:32]	2/5/8/11
SMAC	SMAC0	SMAC[15:0]	0/3/6/9
	SMAC1	SMAC[31:16]	1/4/7/10
	SMAC2	SMAC[47:32]	2/5/8/11
ETHERTYPE	ETHERTYPE	Type/Length	
OTAG_PRI	OTAG	Outer Tag priority/Port-based priority	
DEI_VALUE		DEI	
OTAG_VID		Outer Tag VID/Outer P-VID	
ITAG_PRI	ITAG	Inner Tag priority/Port-based priority	
CFI_VALUE		CFI	
ITAG_VID		Inner Tag VID/Inner P-VID	
IP4_SIP	SIP0	IPv4 source IP[15:0]	0/2/4/6/8/10

	SIP1	IPv4 source IP[31:16]	1/3/5/7/9/11
IP4_DIP	DIP0	IPv4 destination IP[15:0]	0/2/4/6/8/10
	DIP1	IPv4 destination IP[31:16]	1/3/5/7/9/11
IP6_SIP	SIP0	IPv6 SIP [15:0]	0/2/4/6/8/10
	SIP1	IPv6 SIP [31:16]	1/3/5/7/9/11
	SIP2	IPv6 SIP[47:32]	2
	SIP3	IPv6 SIP[63:48]	3
	SIP4	IPv6 SIP[79:64]	4
	SIP5	IPv6 SIP[95:80]	5
	SIP6	IPv6 SIP[111:96]	6
	SIP7	IPv6 SIP[127:112]	7
IP6_DIP	DIP0	IPv6 DIP [15:0]	0/2/4/6/8/10
	DIP1	IPv6 DIP [31:16]	1/3/5/7/9/11
	DIP2	IPv6 DIP[47:32]	2
	DIP3	IPv6 DIP[63:48]	3
	DIP4	IPv6 DIP[79:64]	4
	DIP5	IPv6 DIP[95:80]	5
	DIP6	IPv6 DIP[111:96]	6
	DIP7	IPv6 DIP[127:112]	7
IP_DSCP	IP_TOS_PROTO	IP DSCP	
IP4TOS_IP6TC		IPv4 TOS/IPv6 Traffic Class	
IP4PROTO_IP6NH		IPv4 Protocol/IPv6 Next Header	
ARPOPCODE	IP_TOS_PROTO	ARP/RARP OPCode	
IP_FLAG	IP_FLAG_OFFSET	IPv4 Flags/IPv6 Rsvd+M-Flag	
IP_FRAGMENT_OFFSET		IPv4 Fragment Offset/IPv6 Fragment Offset in Fragment Header	
L4_SRC_PORT	L4_SPORT	TCP/UDP Source Port	
ICMP_CODE	L4_SPORT	ICMP Code/ICMPv6 Code	
ICMP_TYPE		ICMP Type/ICMPv6 Type/IGMP Type	
L4_DST_PORT	L4_DPORT	TCP/UDP Destination Port	
IP_FLAG	L34_HEADER	[1:0]: IPv4 header Flags(bit1 DF, bit0 MF)	
IP6_ESP_HDR_EXIST		IPv6 packet with ESP header	
IP6_AUTH_HDR_EXIST		IPv6 packet with authentication header	
IP6_DEST_HDR_EXIST		IPv6 packet with destination option header	
IP6_FRAG_HDR_EXIST		IPv6 packet with fragment header	
IP6_ROUTING_HDR_EXIST		IPv6 packet with routing header	
IP6_HOP_HDR_EXIST		IPv6 packet with hop-by-hop option header	
IP4_TTL_IP6_HOPLIMIT		IPv4 TTL/IPv6 Hop Limit 2b'00: TTL = 0 2b'01: TTL = 1 2b'10: 2<=TTL<255 2b'11: TTL = 255	
TCP_FLAG		TCP Flag	
ICMP_CODE	ICMP_IGMP	ICMP Code/ICMPv6 Code	
ICMP_TYPE		ICMP Type/ICMPv6 Type	
IGMP_TYPE	ICMP_IGMP	IGMP Type	

TCP_NONZEROSEQ	ICMP_IGMP	TCPNonZeroSequence	
TCP_ECN		TCP ECN ([1]: ECE, [0]: CWR)	
TCP_FLAG		TCP Flag ([5]: URG, [0]: FIN)	
TELNET		TCP & L4 destination port = 23	
SSH		TCP & L4 destination port = 22	
HTTP		TCP & L4 destination port = 80	
HTTPS		TCP & L4 destination port = 443	
SNMP		UDP & L4 destination port = 161/10161	
UNKNOWN_L7		Protocol is TCP/UDP & NOT TELNET/SSH/HTTP/HTTPS/SNMP	
VID_RANGE0	VID_RANGE0	O-VID/I-VID Range Check Mask for range check configuration 0-15	
VID_RANGE1	VID_RANGE1	O-VID/I-VID Range Check Mask for range check configuration 16-31	
IP_RANGE	IP_LEN_RANGE	IPv4/IPv6 Address Range Check Mask	
LEN_RANGE		Packet Length Range Check Mask	
FIELD_SELECTOR_VALID_MSK	FIELD_SELECTOR_VALID	Field Selector Valid Mask	
L2_CRC_ERROR		L2 CRC Error	
IP4_CHKSUM_ERROR		IPv4 checksum Error	
FIELD_SELECTOR0	FIELD_SELECTOR_0	User defined 16-bit field selector 0	
FIELD_SELECTOR1	FIELD_SELECTOR_1	User defined 16-bit field selector 1	
FIELD_SELECTOR2	FIELD_SELECTOR_2	User defined 16-bit field selector 2	
FIELD_SELECTOR3	FIELD_SELECTOR_3	User defined 16-bit field selector 3	
FIELD_SELECTOR4	FIELD_SELECTOR_4	User defined 16-bit field selector 4	
FIELD_SELECTOR5	FIELD_SELECTOR_5	User defined 16-bit field selector 5	
FIELD_SELECTOR6	FIELD_SELECTOR_6	User defined 16-bit field selector 6	2
FIELD_SELECTOR7	FIELD_SELECTOR_7	User defined 16-bit field selector 7	3
FIELD_SELECTOR8	FIELD_SELECTOR_8	User defined 16-bit field selector 8	4
FIELD_SELECTOR9	FIELD_SELECTOR_9	User defined 16-bit field selector 9	5
FIELD_SELECTOR10	FIELD_SELECTOR_10	User defined 16-bit field selector 10	6
FIELD_SELECTOR11	FIELD_SELECTOR_11	User defined 16-bit field selector 11	7
OMPLS_LABEL	OLABEL	Outer MPLS label[15:0]	
	OILABEL	Outer MPLS label[19:16]	
IMPLS_LABEL	ILABEL	Inner MPLS label[15:0]	
	OILABEL	Inner MPLS label[19:16]	
OMPLS_EXP	OLLABEL	Outer EXP	
IMPLS_EXP		Inner EXP	
OMPLS_LABEL_EXIST		Outer label exist	
IMPLS_LABEL_EXIST		Inner label exist	

Fixed field SPN is used to filter a single ingress port while SPM can be used to filter multiple ingress ports by configuring the mask bits.

For ARP/RARP packet, the DMAC field represents the destination hardware address and the DIP field represents the destination protocol address in ARP/RARP header.
For ARP/RARP packet, the SMAC field represents the source hardware address and the SIP field represents the source protocol address in ARP/RARP header.

OTAG_VID and ITAG_VID qualify VID field in tag for tagged packet or port-based priority for priority-tagged and untagged packet.

OTAG_PRI and ITAG_PRI qualify Priority field in tag for tagged and priority-tagged packet or port-based priority for untagged packet.

The value of ETHERTYPE can follow with CPU Tag、Outer Tag、Inner Tag or the first Type/Length value follows with RFC_1042 header(AA-AA-03-00-00-00)。

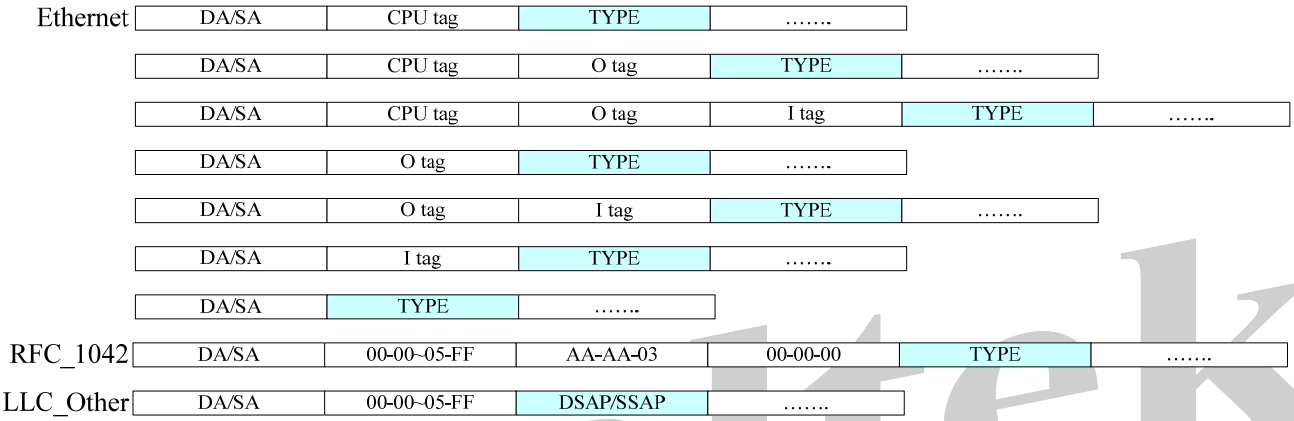


Figure 18: Ether Type Field Position

In default, L2 CRC and IPv4 checksum error packets aren't processed by ACL. For qualifying L2_CRC_ERROR and IP4_CHKSUM_ERROR, there is a register field ACL_CTRL.CRC_LOOKUP_EN to enable ACL for L2 CRC and IPv4 checksum error packets.

The ICMP_IGMP represents the ICMP/ICMPv6 code, type for ICMP/ICMPv6 packet and represents IGMP type for IGMP packet and represents TCP ECN, TCP Flag, and management applications for TCP/UDP packet.

IP4PROTO_IP6NH is from Protocol field for an IPv4 packet and from the last known Next Header if there are multiple Next Headers appeared for an IPv6 packet.

FIELD_SELECTOR 6-11 are defined as combo fields for supporting IPv6 SIP. They are interpreted as IPv6 SIP [127:32] if the received packet is an IPv6 packet. Otherwise, they are interpreted as normal field selectors. Thus, if the field selectors would be applied on both IPv4 and IPv6 traffic, then must use FIELD_SELECTOR 0-5 instead.

API REFERENCE

```
rtk_acl_ruleEntryField_read(uint32 unit, rtk_acl_phase_t phase, rtk_acl_id_t entry_idx, rtk_acl_fieldType_t
type, uint8 *pData, uint8 *pMask)
rtk_acl_ruleEntryField_write(uint32 unit, rtk_acl_phase_t phase, rtk_acl_id_t entry_idx, rtk_acl_fieldType_t
type, uint8 *pData, uint8 *pMask)

rtk_acl_ruleEntryField_get(
    uint32                  unit,
    rtk_acl_phase_t         phase,
    rtk_acl_id_t            entry_idx,
    uint8                  *pEntry_buffer,
    rtk_acl_fieldType_t     type,
    uint8                  *pData,
    uint8                  *pMask)
rtk_acl_ruleEntryField_set(
    uint32                  unit,
    rtk_acl_phase_t         phase,
    rtk_acl_id_t            entry_idx,
    uint8                  *pEntry_buffer,
    rtk_acl_fieldType_t     type,
```

```
    uint8           *pData,
    uint8           *pMask)
```

5.4.3 Egress Field Type

The following field types are only applicable for Egress ACL.

Table 49: Egress ACL Field Type and Template Field Mapping Table

Field Type (USER_FIELD_XXX)	Template Field (TMPLTE_FIELD_XXX)	Description	Template Field Position Limitation
DPMM	DPMMASK	Comparison result of destination port bitmap mask configurations	
DPM	DPM0	Final destination Port Mask [15:0] *Port Mask after egress filtering	0/4/8
	DPM1	Final destination Port Mask [31:16] *Port Mask after egress filtering	1/5/9
	DPM2	Final destination Port Mask [47:32] *Port Mask after egress filtering	2/6/10
	DPM3	Final destination Port Mask [52:48] *Port Mask after egress filtering	3/7/11
DPN	DPM3	Final destination port number *Port number after egress filtering	3/7/11
UCAST_DA		Unicast DA	
MCAST_DA		Multicast DA *Includes L2/IP4/IP6 multicast but excludes broadcast	
SA_LUT_RESULT		SA Lookup Result *0: lookup miss 1: lookup hit	
DA_LUT_RESULT		DA Lookup Result *0: lookup miss 1: lookup hit	
NONZERO_DPM		Non-Zero DPM *Packet has at least one egress port	
L2_DPM	L2DPM0	L2 lookup destination Port Mask [15:0] *Port Mask given by L2 lookup which before egress filtering	0/4/8
	L2DPM1	L2 lookup destination Port Mask [31:16] *Port Mask given by L2 lookup which before egress filtering	1/5/9
	L2DPM2	L2 lookup destination Port Mask [47:32] *Port Mask given by L2 lookup which before egress filtering	2/6/10
	L2DPM3	L2 lookup destination Port Mask [52:48] *Port Mask given by L2 lookup which before egress filtering	3/7/11
L2_DPN	L2DPM3	L2 lookup destination port number *Port number given by L2 lookup which before egress filtering	3/7/11
ATTACK		Attack Packet	
DP		Drop Precedence	

INT_PRI	IVLAN	Internal Priority	
IVID		Ingress Inner VID	
OVID	OVLAN	Ingress Outer VID	
FWD_VID	FWD_VID	Forwarding VID *Refer to VLAN Developer Guide	
IGR_ACL_DROP_HIT		Ingress ACL drop action hit	
IGR_ACL_COPY_HIT		Ingress ACL copy action hit	
IGR_ACL_REDIRECT_HIT		Ingress ACL redirect action hit	
IGR_ACL_ROUTING_HIT		Ingress ACL unicast routing action hit	

DPN/L2DPN is used to filter a single egress port while DPM/L2DPM can be used to filter multiple egress ports by configuring the mask bits. The difference between DPN(DPM) and L2DPN(L2DPM) is that L2DPN is the destination port directly given by L2 lookup while DPN is the final destination port which further processed by egress filtering modules, such as traffic isolation, egress VLAN filtering, egress Spanning Tree filtering, and so on.

INT_PRI is the internal priority determined by Priority Decision module; please refer to Priority Decision Developer Guide for the detail.

IVID/OVID is the VID determined by ingress inner/outer VLAN ID decision flow; please refer to Ingress Inner/Outer VLAN ID Decision Flowchart of VLAN Developer Guide for the detail.

One thing needs to be awarded is that a template is not dedicated create for Ingress or Egress ACL. It depends on the chosen Field Type to be applicable for Ingress or Egress ACL. The template that contains the Egress ACL field type should not be mapped by Ingress ACL block.

API REFERENCE

```
rtk_acl_ruleEntryField_read(uint32 unit, rtk_acl_phase_t phase, rtk_acl_id_t entry_idx, rtk_acl_fieldType_t
type, uint8 *pData, uint8 *pMask)
rtk_acl_ruleEntryField_write(uint32 unit, rtk_acl_phase_t phase, rtk_acl_id_t entry_idx, rtk_acl_fieldType_t
type, uint8 *pData, uint8 *pMask)

rtk_acl_ruleEntryField_get(
    uint32                  unit,
    rtk_acl_phase_t         phase,
    rtk_acl_id_t            entry_idx,
    uint8                   *pEntry_buffer,
    rtk_acl_fieldType_t     type,
    uint8                   *pData,
    uint8                   *pMask)
rtk_acl_ruleEntryField_set(
    uint32                  unit,
    rtk_acl_phase_t         phase,
    rtk_acl_id_t            entry_idx,
    uint8                   *pEntry_buffer,
    rtk_acl_fieldType_t     type,
    uint8                   *pData,
    uint8                   *pMask)
```

5.4.4 Source and Destination Port Bitmap Mask

The device supports 16 source port bitmap configurations (RNG_CHK_SPM_CTRL) and each configuration contains a source port bitmap.
The source port of received packet is used to compare with 16 source port bitmap configurations simultaneously, and the 16 bits comparison result (each bit represents a comparison result of a source port bitmap configuration)

can be examined by filed type SPMM. The comparison result is true if the source port is in the configured bitmap.

Table 50: RNG_CHK_SPM_CTRL Register

Field Name	Bits	Description
SPM_1	21	Source port bitmap represents port 32 to 52
SPM_0	32	Source port bitmap represents port 0 to 31

The device supports 16 destination port bitmap configurations (RNG_CHK_DPM_CTRL) and each configuration contains a destination port bitmap.
The destination port of received packet is used to compare with 16 destination port bitmap configurations simultaneously, and the 16 bits comparison result (each bit represents a comparison result of a destination port bitmap configuration) can be examined by filed type DPMM. The comparison result is true if the destination port is in the configured bitmap.

Table 51: RNG_CHK_DPM_CTRL Register

Field Name	Bits	Description
DPM_1	21	Destination port bitmap represents port 32 to 52
DPM_0	32	Destination port bitmap represents port 0 to 31

Source and destination port bitmap configurations reduce the used template fields from four (SPM0-3/DPM0-3) to one (SPMM/DPMM) to filter multiple source/destination ports.

API REFERENCE

*int32 rtk_acl_rangeCheckSrcPort_get(uint32 unit, uint32 index, rtk_acl_rangeCheck_portMask_t *pData)*
*int32 rtk_acl_rangeCheckSrcPort_set(uint32 unit, uint32 index, rtk_acl_rangeCheck_portMask_t *pData)*

*int32 rtk_acl_rangeCheckDstPort_get(uint32 unit, uint32 index, rtk_acl_rangeCheck_portMask_t *pData)*
*int32 rtk_acl_rangeCheckDstPort_set(uint32 unit, uint32 index, rtk_acl_rangeCheck_portMask_t *pData)*

5.4.5 Range Check

The device supports four kinds of range check function: VLAN, IP, L4 port, and packet length to filter a packet within specific range.

The device supports 32 VID range check entries. For priority-tag and untag packet, the device takes port-based VID to process. The 32-bits result of VID range check is examined by the VIDRANGE0 and VIDRANGE1 field types.

Table 52: RNG_CHK_VID_CTRL Register

Field Name	Bits	Description
VID_UPPER	12	VID upper bound
VID_LOWER	12	VID lower bound
REVERSE	1	Reverse the filtering result 1'b0: don't reverse 1'b1: reverse
TYPE	1	1'b0: inner VID 1'b1: outer VID

The device supports 8 IP range check entries which can be used as 8 IPv4 or 2 IPv6 or 4 IPv6 suffix range check. The 8-bits result of IP range check is examined by IP_RANGE field types.

Table 53: RNG_CHK_IP_CTRL Register

Field Name	Bits	Description
REVERSE	1	Reverse the filtering result 1'b0: don't reverse 1'b1: reverse
TYPE	3	3'h0: IPv4 SIP 3'h1: IPv4 DIP 3'h2: IPv6 SIP 3'h3: IPv6 DIP 3'h4: IPv6 SIP [63:0] suffix 3'h5: IPv6 DIP [63:0] suffix 3'h6~7: reserved

Table 54: RNG_CHK_IP_RNG Register

Field Name	Bits	Description
IP_UPPER	32	IP address upper bound
IP_LOWER	32	IP address lower bound

For IPv6 range check, entry 0/4 represents SIP/DIP [31:0], entry 1/5 represents SIP/DIP [63:32], entry 2/6 represents SIP/DIP [95:64], and entry 3/7 represents SIP/DIP [127:96]. The device compares each 32-bits individually for IPv6 range check.

For IPv6 suffix range check, entry 0/2/4/6 represents SIP/DIP [31:0] and entry 1/3/5/7 represents SIP/DIP [63:32]. When 2n and (2n+1) entry of IP range check are configured to the same IPv6 suffix type, the device compares 64-bits altogether and the result is reflected to entry 2n (result of entry 2n+1 is cleared to 0).
The IPv6 bit notation is defined as: *2001[127:112]:0db8:1f70:2633:0999:0de8:7648:06e8[15:0]*

The device supports 8 L4 port range check entries. The 8-bits result of L4 port range check is examined by PORT_RANGE field type.

Table 55: RNG_CHK_L4PORT_CTRL Register

Field Name	Bits	Description
REVERSE	1	Reverse the filtering result 1'b0: don't reverse 1'b1: reverse
TYPE	1	1'b0: TCP/UDP source port 1'b1: TCP/UDP destination port

Table 56: RNG_CHK_L4PORT_RNG Register

Field Name	Bits	Description
L4PORT_UPPER	16	TCP/UDP port upper bound
L4PORT_LOWER	16	TCP/UDP port lower bound

The device supports 8 packet length range check entries. The length includes 4-bytes CRC field. The result of packet length range check is examined by LEN_RANGE field type.

Table 57: RNG_CHK_PKT_LEN_CTRL Register

Field Name	Bits	Description
PKTLEN_UPPER	14	Packet length upper bound
PKTLEN_LOWER	14	Packet length lower bound
REVERSE	1	Reverse the filtering result 1'b0: don't reverse 1'b1: reverse

Range check result is valid only if the packet format is complied with the range check data type.

API REFERENCE

```
int32 rtk_acl_rangeCheckVid_get(uint32 unit, uint32 index, rtk_acl_rangeCheck_vid_t *pData)
int32 rtk_acl_rangeCheckVid_set(uint32 unit, uint32 index, rtk_acl_rangeCheck_vid_t *pData)

int32 rtk_acl_rangeCheckIp_get(uint32 unit, uint32 index, rtk_acl_rangeCheck_ip_t *pData)
int32 rtk_acl_rangeCheckIp_set(uint32 unit, uint32 index, rtk_acl_rangeCheck_ip_t *pData)

int32 rtk_acl_rangeCheckL4Port_get(uint32 unit, uint32 index, rtk_acl_rangeCheck_l4Port_t *pData)
int32 rtk_acl_rangeCheckL4Port_set(uint32 unit, uint32 index, rtk_acl_rangeCheck_l4Port_t *pData)

int32 rtk_acl_rangeCheckPacketLen_get(uint32 unit, uint32 index, rtk_acl_rangeCheck_packetLen_t
*pData)
int32 rtk_acl_rangeCheckPacketLen_set(uint32 unit, uint32 index, rtk_acl_rangeCheck_packetLen_t
*pData)
```

5.4.6 Field Selector

The device supports 12 set of 16-bits field selectors which are used as user defined fields.
There are 8 kinds of start positions to support L2~L4 applications. Each field selector entry is composed of start position and offset.

Table 58: PARSER_FIELD_SELTOR_CTRL Register

Field Name	Bits	Description
OFFSET	8	Offset in bytes.
FMT	3	Define the start address for offset. 3'h0: Raw packet (start at DA field) 3'h1: LLC packet (start at length field) 3'h2: L3 packet (start after Ether type field) 3'h3: ARP/RARP packet (start at ARP/RARP header) 3'h4: IPv4 packet (start at IPv4 header) 3'h5: IPv6 packet (start at IPv6 header) 3'h6: IP payload (start at IP payload) 3'h7: L4 payload (start after TCP/UDP header, only TCP/UDP packet is supported.)

Field selector doesn't support to locate the VLAN Tag and only allows user to select any 2 bytes within the first 180 bytes of the packet. **Field selector 6-11 will be IPv6 SIP [127:32] when the receiving packet is IPv6**.

API REFERENCE

```
int32 rtk_acl_fieldSelector_get(uint32 unit, uint32 fs_idx, rtk_acl_fieldSelector_data_t *pFs)
int32 rtk_acl_fieldSelector_set(uint32 unit, uint32 fs_idx, rtk_acl_fieldSelector_data_t *pFs)
```

5.4.7 Pre-defined Template

The device supports 5 pre-defined templates. They are assigned with template ID 0-4 while 5-7 are for user defined templates. The pre-defined templates are shown as below:

Table 59: Pre-defined Template 0

Field 0	Field 1	Field 2	Field 3	Field 4	Field 5
SPM0	SPM1	OTAG/ITAG	SMAC0	SMAC1	SMAC2
Field 6	Field 7	Field 8	Field 9	Field 10	Field 11
DMAC0	DMAC1	DMAC2	ETHER TYPE	SPM2	SPM3

This template is for MAC ACL applications.

Table 60: Pre-defined Template 1

Field 0	Field 1	Field 2	Field 3	Field 4	Field 5
SIP0	SIP1	DIP0	DIP1	IPTOSPROTO	L4SPORT
Field 6	Field 7	Field 8	Field 9	Field 10	Field 11
L4DPORT	ICMPIGMP	SPM0	SPM1	SPM2	SPM3

This template is for IPv4 ACL applications.

Table 61: Pre-defined Template 2

Field 0	Field 1	Field 2	Field 3	Field 4	Field 5
DMAC0	DMAC1	DMAC2	ITAG	ETHER TYPE	IPTOSPROTO
Field 6	Field 7	Field 8	Field 9	Field 10	Field 11
L4DPORT	L4SPORT	SIP0	SIP1	DIP0	DIP1

This template is for STP/LACP/LLDP/GVRP/PPPoE/DHCP/QoS applications.

Table 62: Pre-defined Template 3

Field 0	Field 1	Field 2	Field 3	Field 4	Field 5
DIP0	DIP1	DIP2	DIP3	DIP4	DIP5
Field 6	Field 7	Field 8	Field 9	Field 10	Field 11
DIP6	DIP7	L4DPORT	L4SPORT	ICMPIGMP	IPTOSPROTO

This template is for L3&L4 IPv6 ACL applications.

Table 63: Pre-defined Template 4

Field 0	Field 1	Field 2	Field 3	Field 4	Field 5
SIP0	SIP1	SIP2	SIP3	SIP4	SIP5
Field 6	Field 7	Field 8	Field 9	Field 10	Field 11
SIP6	SIP7	SPM0	SPM1	SPM2	SPM3

This template is for IPV6 ACL.

API REFERENCE

```
rtk_acl_template_get(uint32 unit, uint32 template_id, rtk_acl_template_t *pTemplate)
rtk_acl_template_set(uint32 unit, uint32 template_id, rtk_acl_template_t *pTemplate)
```

5.5 ACL Operation

Each ACL entry supports a REVERSE operation which is used to reverse the compare result.

To support a compare key larger than length of a single template (192-bits), the device supports AGGREGATION operations. The length of compare key can be double length (384-bits) by enabling AGGREGATION_1 or AGGREGATION_2 operation and quadruple length (768 bits) by enabling both AGGREGATION_1 and AGGREGATION_2 operations.

The AGGREGATION_1 operation is used to aggregate the result of two consecutive (0 & 1, 2 & 3, …, 2n & 2n+1) entries in the same block as one. The device executes the logical AND operation for these two aggregated entries and updates the result back to the entry with lower index and clear the result of the entry with higher index to 0. Each

index 2N entry supports an AGGREGATION_1 operation and it makes sense if the two aggregated entries maps to different templates.

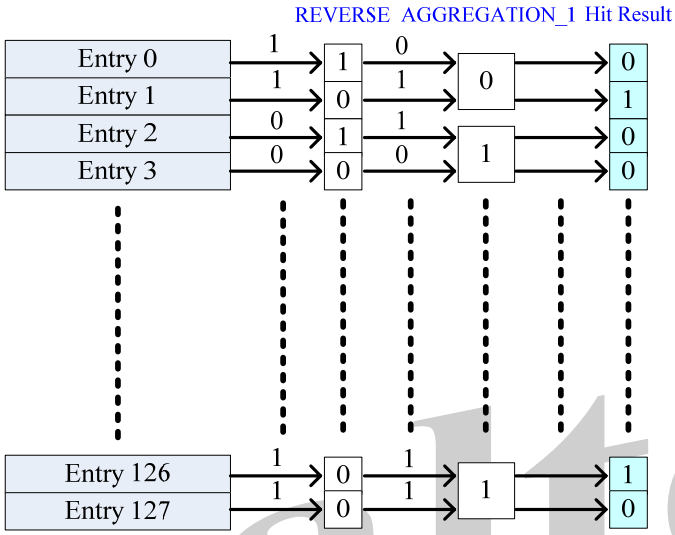


Figure 19: REVERSE and AGGREGATION_1 Operation Flow

The AGGREGATION_2 operation is used to aggregate the result of two entries in the two consecutive blocks as one. The device executes the logical AND operation for these two aggregated entries and updates the result back to the entry in the lower block and clear the result of the entry in the higher block to 0. Each index 2N+256M (where N,M = 0,1,2...) entry supports an AGGREGATION_2 operation and it makes sense if the two aggregated entries maps to different templates.

When AND1 and AND2 operations are both enabled, the length of compare key is up to 768 bits.

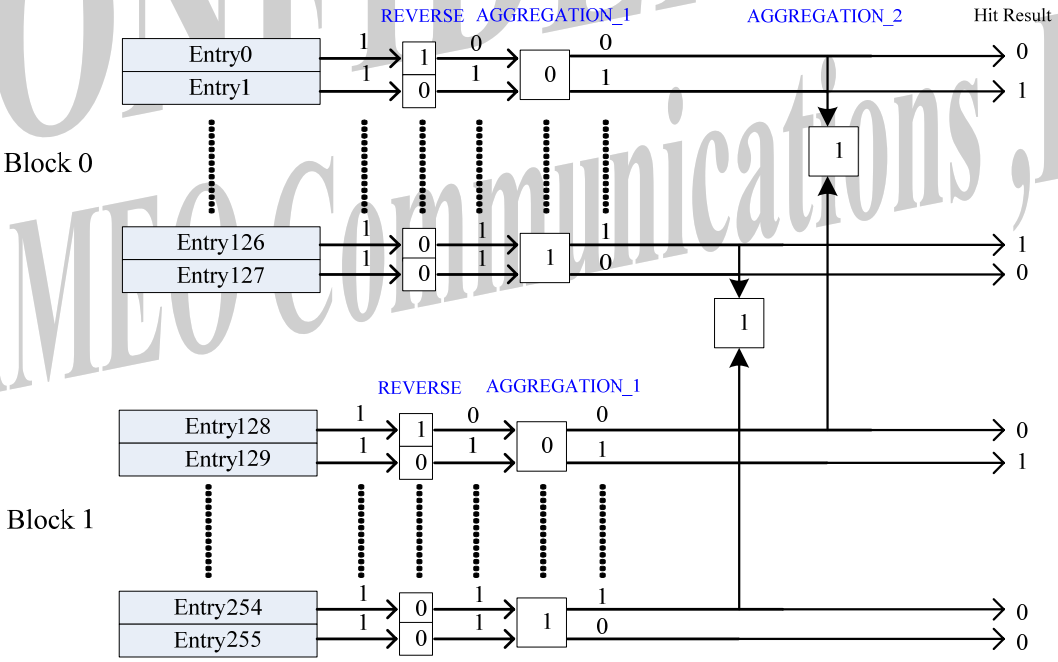


Figure 20: REVERSE, AGGREGATION_1 and AGGREGATION_2 Operation Flow

API REFERENCE

```
int32 rtk_acl_ruleOperation_get(uint32 unit, rtk_acl_phase_t phase, rtk_acl_id_t entry_idx,
    rtk_acl_operation_t *pOperation)
int32 rtk_acl_ruleOperation_set(uint32 unit, rtk_acl_phase_t phase, rtk_acl_id_t entry_idx,
    rtk_acl_operation_t *pOperation)
```

5.6 ACL Action

Each ACL entry has an action mask which is used to indicate the actions to be executed when the entry is hit.
Ingress ACL supports up to 9 kind of actions which can be executed concurrently.
Egress ACL also supports up to 9 kind of actions which can be executed concurrently.

5.6.1 Ingress ACL Action

Ingress ACL action mask:

- Forward
- Statistic
- Mirror
- Meter
- Ingress Inner VLAN Assignment
- Ingress Outer VLAN Assignment
- Priority Assignment
- MPLS
- Bypass

1.6.1.1 Forward Action

Ingress Forward action provides below configurations:

- Permit
- Drop
- Copy packet to single port
- Copy packet to multiple ports
- Redirect packet to single port
- Redirect packet to multiple ports
- Unicast Routing

The Permit action can be used to avoid a specific flow from hitting other Forward actions. For instance, if traffic comes from subnet 192.168.1.X would be dropped except from host 192.168.1.1, a Permit entry should be inserted to the index lower than the Drop entry to grant the traffic from 192.168.1.1. The Permit action only takes effect within the ACL module.

The Copy action adds one or more egress ports to the original egress port list. For Copy to CPU applications, such as ARP, DHCP, just specify the CPU Port ID as the copied port. The packet copied to CPU has higher priority than traffic isolation, egress VLAN filtering, egress Spanning Tree filtering, lookup miss action and lookup miss flooding domains.

The Redirect action overrides the original egress port list. For Trap to CPU applications, just specify the CPU Port ID as the redirected port. The packet trapped to CPU has higher priority than traffic isolation, egress VLAN filtering, egress Spanning Tree filtering, lookup miss action and lookup miss flooding domains.

Regarding the unicast routing, please refer to the Static Routing Developer Guide.

1.6.1.2 Statistic Action

Statistic action provides two configurations:

- 32-bits packet-based counter
- 64-bits byte-based counter

The Statistic action specifies the counter type and an index to the Counter. The device supports 1K counters and each counter can be used as one 64-bits byte-based counter or two 32-bits packet-based counters which is specified by Statistic action. Thus, up to 2K 32-bits packet-based or 1K 64-bits byte-based counters can be supported.

If counter type is 32-bits packet-based counter, the counter is increased with the number of packets. If counter type is 64-bits byte-based counter, the counter is increased with the packet length.

The 1K counters are divided into 16 counter blocks, that is, each counter block contains 64 counters. Unlike other actions, per ACL block can execute a statistic action but the specified (indexed) counters must reside in different counter blocks. This is because each counter block can only execute a counter increment for a packet. Thus, if a counter resided in counter block Y is indexed by an ACL entry resided in block X, ACL entry other than in block X shouldn't index to a counter resided in counter block Y. Under such limitation, up to 16 counters which are all resided in different counter blocks can be increased concurrently for a packet.

There is another limitation that a counter block can't be used by Ingress and Egress ACL blocks concurrently.

1.6.1.3 Mirror Action

The Mirror action specifies mirror entry index for retrieving the mirroring port. Other configurations of the indexed mirror entry are not utilized.
The action is considered as ingress mirror, so the packet mirrored is the original packet content.

1.6.1.4 Meter Action

The Meter action specifies the meter index for retrieving meter information. The device supports 512 meters which is divided into 16 blocks. Each meter block contains 32 meters. Unlike other actions, per ACL block can execute a meter action but the specified (indexed) meters must reside in different meter blocks. This is because each meter block can only execute a meter for a packet. Thus, if a meter resided in meter block Y is indexed by an ACL entry resided in block X, ACL entry other than in block X shouldn't index to a meter resided in meter block Y. Under such limitation, up to 16 meters which are all resided in different meter blocks can be executed concurrently for a packet to support the Hierarchical Policing.

The packet gets drop or marked a color according to the meter configuration if traffic rate is over than the meter rate. Because multiple meters can be executed for a packet, the priority for meter action is Drop > Red Coloring > Yellow Coloring.

Regarding meter configurations, please refer to Meter (srTCM/trTCM) Developer Guide.

1.6.1.5 Ingress Inner VLAN Assignment Action

Ingress Inner VLAN assignment action provides below configurations:

- Assign inner VLAN
- Shift inner VLAN
- Assign inner VLAN by shifting outer VLAN

Assign inner VLAN action directly assigns an inner VLAN to the qualified packet.

Shift inner VLAN action assigns an inner VLAN to the qualified packet by shifting packet's inner VID with a specific value. The action is useful for the ingress VLAN translation application that shares the same translation offset. If VLAN 1~100 come from downlink port would be translated to VLAN 1001~1100 and forward to uplink port, only one ACL entry is needed. If assigned VLAN (the VLAN after shifting) is larger than 4095, then it is wrapped around. For instance, Packet's VID=4000 and shift value=100, then the assigned VLAN will be 4 (4000+100-4096).
Shift inner VLAN action only takes effect if the receiving packet is an inner tag packet.

Assign inner VLAN by shifting outer VLAN action is similar to Shift inner VLAN action except it assigns an inner VLAN to the qualified packet by shifting packet's outer VID with a specific value.
Assign inner VLAN by shifting outer VLAN action only takes effect if the receiving packet is an outer tag packet.

1.6.1.6 Ingress Outer VLAN Assignment Action

Ingress Outer VLAN assignment action provides below configurations:
- Assign outer VLAN
- Shift outer VLAN
- Assign outer VLAN by shifting inner VLAN

The behavior of Ingress Outer VLAN assignment action is similar to Ingress Inner VLAN assignment action.

1.6.1.7 Priority Assignment Action

Priority action assigns an internal priority to ingress packet. The assigned priority is one of the internal priority sources. The final internal priority which for mapping egress queue is made by Priority Decision module. Regarding Priority Decision module, please refer to Priority Decision Developer Guide.

1.6.1.8 MPLS Action

MPLS action provides two configurations:
- Push single label
- Push double label

The MPLS action specifies the labeling type and an index to the LIB (Label Information Base) table for retrieving the label information. The device supports 512 LIB entries and each LIB entry contains two labels. For Push double label action, both labels of the LIB entry are used. For Push single label action, only a label of the LIB entry is used. Thus, up to 512 single label or 256 double labels can be supported.

1.6.1.9 Bypass Action

Bypass action provides below configurations:
- Bypass Ingress Bandwidth Control and Storm Control filtering
- Bypass Ingress Spanning Tree filtering
- Bypass Ingress and Lookup filtering

When Bypass Ingress Bandwidth Control and Storm Control filtering action is enabled, the packet will not be filtered by Ingress Bandwidth Control and Storm Control modules. The action prevents qualified traffic from dropping by Ingress Bandwidth Control and Storm Control modules. The bypassed traffic doesn't consume the bandwidth of Ingress Bandwidth Control and Storm Control modules.

When Bypass Ingress Spanning Tree filtering action is enabled, the packet will not be filtered by Ingress Spanning Tree module. The action prevents qualified traffic from dropping by Ingress Spanning Tree module.

When Bypass Ingress and Lookup filtering action is enabled, the packet will not be filtered by VLAN, Spanning Tree, RMA, Attack Prevention, OAM, Storm Control, L2 modules.

Three Bypass actions can be enabled concurrently.

API REFERENCE

```
int32 rtk_acl_ruleAction_get(
    uint32                  unit,
    rtk_acl_phase_t         phase,
    rtk_acl_id_t            entry_idx,
    rtk_acl_action_t        *pAction)
int32 rtk_acl_ruleAction_set(
    uint32                  unit,
    rtk_acl_phase_t         phase,
    rtk_acl_id_t            entry_idx,
```

```
    rtk_acl_action_t        *pAction)
```

5.6.2 Egress ACL Action

Egress ACL action mask:
- Forward
- Statistic
- Mirror
- Meter
- Egress Inner VLAN Assignment
- Egress Outer VLAN Assignment
- Priority Assignment
- Remarking
- Egress Tag Status

1.6.2.1 Forward Action

Egress Forward action provides below configurations:
- Permit
- Drop
- Copy packet to single port
- Copy packet to multiple ports
- Redirect packet to single port
- Redirect packet to multiple ports
- Filter

The actions other than Filter are similar to Ingress Forward Action.
The Filter action masks the egress port list with the specified port mask, that is, doing logical AND operation for egress port list and specified port mask. One of the possible applications is flow-based port isolation.

1.6.2.2 Statistic Action

Please refer to Ingress Statistic Action.

1.6.2.3 Mirror Action

The Egress Mirror action is similar to Ingress Mirror Action. But the Egress Mirror action can further specify to mirror original or modified packet. For instance, it can specify to mirror the packet before or after doing VLAN translation for a VLAN translation packet.

1.6.2.4 Meter Action

Please refer to Ingress Meter Action.

1.6.2.5 Egress Inner VLAN Assignment Action

Egress Inner VLAN assignment action provides below configurations:
- Assign inner VLAN
- Shift inner VLAN
- Assign inner VLAN by shifting outer VLAN
- Assign inner VLAN by copying outer VLAN

Please refer to Ingress Inner VLAN Assignment Action.
The action can also assign the Egress Inner TPID by specifying the Egress Inner TPID index.

The difference between Ingress and Egress Inner VLAN Assignment is that only VLAN assigned by Ingress Inner VLAN Assignment participates the Forwarding VLAN decision.

1.6.2.6 Egress Outer VLAN Assignment Action

Egress Outer VLAN assignment action provides below configurations:
- Assign outer VLAN
- Shift outer VLAN
- Assign outer VLAN by shifting inner VLAN
- Assign outer VLAN by copying inner VLAN

Please refer to Ingress Outer VLAN Assignment Action.
The action can also assign the Egress Outer TPID by specifying the Egress Outer TPID index.

The difference between Ingress and Egress Outer VLAN Assignment is that only VLAN assigned by Ingress Outer VLAN Assignment participates the Forwarding VLAN decision.

1.6.2.7 Priority Assignment Action

Egress Priority assignment action provides two configurations:
- Common priority assignment
- CPU priority assignment

Common priority assignment action assigns final internal priority to ingress packet regardless of egress port list which overrides the internal priority made by Priority Decision module. The final internal priority is then used to map egress queue.

CPU priority assignment action ONLY assigns final internal priority to the packet forwarded to CPU port. If the egress port list of a certain packet contains CPU port, then the packet forwards to CPU port can have different priority than the packet forwards to normal ports. For instance, the packet (ex: ARP, DHCP) which is copied to CPU has egress port list 1,3,5 and CPU port, the packet forwards to CPU port is assigned a priority by CPU priority assignment action while packet forwards to 1,3,5 still assigned a priority by Priority Decision module as usual.

Regarding Priority Decision module, please refer to Priority Decision Developer Guide.

1.6.2.8 Remarking Action

Remarking action provides below configurations:
- Remark inner tag
- Remark outer tag
- Remark DSCP
- Remark IP Precedence
- Remark inner tag by copying outer tag
- Remark outer tag by copying inner tag

Remark DSCP action takes effect only if receiving packet is an IPv4/IPv6 packet. For IPv6 packet, DSCP is resided in Traffic Class field.

Remark IP Precedence action takes effect only if receiving packet is an IPv4 packet.

Remark inner tag by copying outer tag action remarks inner tag by copying priority from outer tag, thus, the action takes effect only if receiving packet is an outer tag packet.

Remark outer tag by copying inner tag action remarks outer tag by copying priority from inner tag, thus, the action takes effect only if receiving packet is an inner tag packet.

Remark inner tag by copying outer tag action and Remark outer tag by copying inner tag action may be used in

VLAN translation or Q-in-Q applications.

1.6.2.9 Egress Tag Status Action

Egress tag status action specifies the inner and outer tag status for an outgoing packet. Four statuses can be specified: untag, tag, keep content, NOP. Keep content keeps tag format and tag content, that is, untag in then untag out, priority tag in then priority tag out, tag in then tag out, and the content (VID/Priority/CFI) is all kept. NOP represents NULL operation which doesn't specify the tag status. NOP is used if user just wants to specify only inner or outer tag status.

If egress inner/outer tag status is specified to be keep content, it overrides the egress inner/outer VLAN assignment action.

API REFERENCE

```
int32 rtk_acl_ruleAction_get(
    uint32                  unit,
    rtk_acl_phase_t         phase,
    rtk_acl_id_t            entry_idx,
    rtk_acl_action_t        *pAction)
int32 rtk_acl_ruleAction_set(
    uint32                  unit,
    rtk_acl_phase_t         phase,
    rtk_acl_id_t            entry_idx,
    rtk_acl_action_t        *pAction)
```

5.7 Block Operation

There are two kinds of block operations supported by the device. One operation is to enable the multiple hit function for a ACL block while the other operation is to group the physical ACL blocks as a logical block. They are described in following sections.

5.7.1 Multiple Hit

The device supports a multiple hit register ACL_BLK_RESULT_CTRL to specify the multiple hit status for each block. If an ACL block is configured to single hit, only the entry with the lowest index is marked as hit. If an ACL block is configured to multiple hit, more than one entry can be marked as hit.

Multiple hit is useful to qualify different packet characteristics within a block to benefit the ACL entry arrangement.

Table 64: ACL_BLK_RESULT_CTRL Register

Field Name	Bits	Description
BLK_RESULT_MULTI	1	ACL block hit configuration. 1'b0: single hit for a block 1'b1: multiple hit for a block

API REFERENCE

```
int32 rtk_acl_blockResultMode_get(uint32 unit, uint32 block_idx, rtk_acl_blockResultMode_t *pMode)
int32 rtk_acl_blockResultMode_set(uint32 unit, uint32 block_idx, rtk_acl_blockResultMode_t mode)
```

5.7.2 Logical Block Grouping

For some specific applications, 128 entries of a block may not be enough. The device supports block grouping function ACL_BLK_GROUP_CTRL for grouping physical blocks to a logical block. Each two adjacent blocks

(2N,2N+1) have a BLK_GROUP_2 register configuration. Each four adjacent blocks (4N~4N+3) have a BLK_GROUP_4 register configuration. Each eight adjacent blocks (8N~8N+7) have a BLK_GROUP_8 register configuration. And one BLK_GROUP_ALL register configuration to group all blocks.

A logical block only outputs a single hit entry which is the lowest index entry among hit entries. Grouping Ingress and Egress ACL blocks together is allowed even the grouping doesn't make sense.

Table 65: ACL_BLK_GROUP_CTRL Register

Field Name	Bits	Description
BLK_GROUP_ALL	1	Group all blocks as a logic block 1'b0: don't group 1'b1: group
BLK_GROUP_8	2	Group 8 adjacent blocks (8N~8N+7) as a logic block 1'b0: don't group 1'b1: group
BLK_GROUP_4	4	Group 4 adjacent blocks (4N~4N+3) as a logic block 1'b0: don't group 1'b1: group
BLK_GROUP_2	9	Group 2 adjacent blocks (2N,2N+1) as a logic block 1'b0: don't group 1'b1: group

Below diagram shows the block grouping sequence:

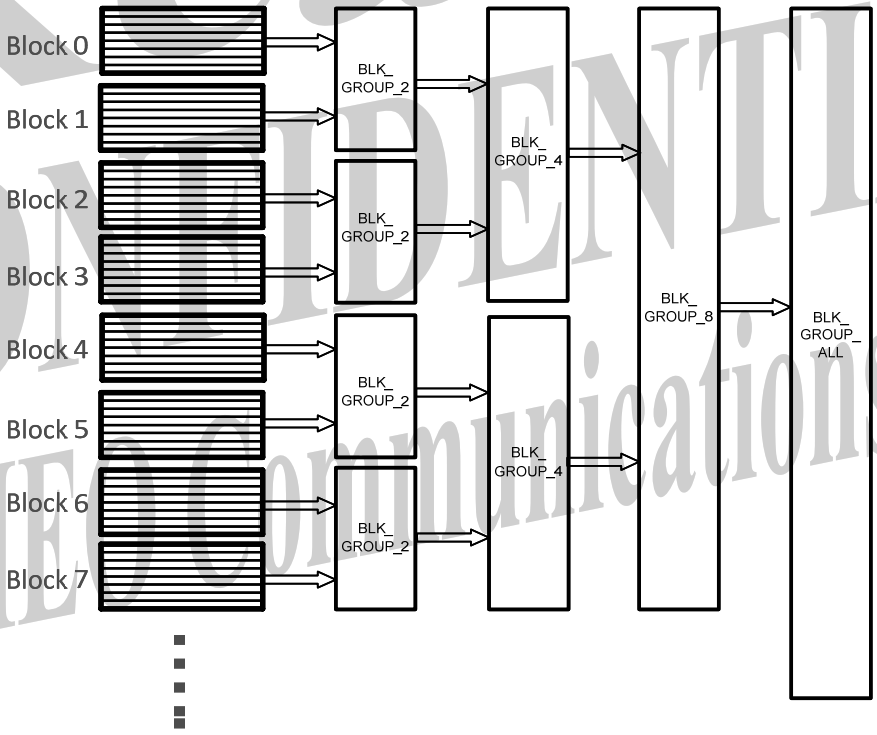


Figure 21: Logical Block Grouping Flow

API REFERENCE

```
int32 rtk_acl_blockGroupEnable_get(uint32 unit, uint32 block_idx, rtk_acl_blockGroup_t group_type,
    rtk_enable_t *pEnable)
int32 rtk_acl_blockGroupEnable_set(uint32 unit, uint32 block_idx, rtk_acl_blockGroup_t group_type,
    rtk_enable_t enable)
```

5.8 ACL Hit Indication

Each ACL entry has a hit indication register field ACL_RULE_HIT_INDICATION.RULE_INDICATION to indicate the hit status. The hit status is the hit result after executing ACL operations, Multiple Hit and Logical Block Grouping.

API REFERENCE

```
int32 rtk_acl_ruleHitIndication_get(
    uint32                  unit,
    rtk_acl_phase_t         phase,
    rtk_acl_id_t            entry_idx,
    uint32                  reset,
    uint32                  *pIsHit)
```

5.9 Action Arbitration and Execution

A packet may hit multiple ACL entries concurrently, thus, action arbitration for the hit entries is made before action execution. For the hit entries, the device per action group picks up the lowest index entries among blocks to execute the actions.

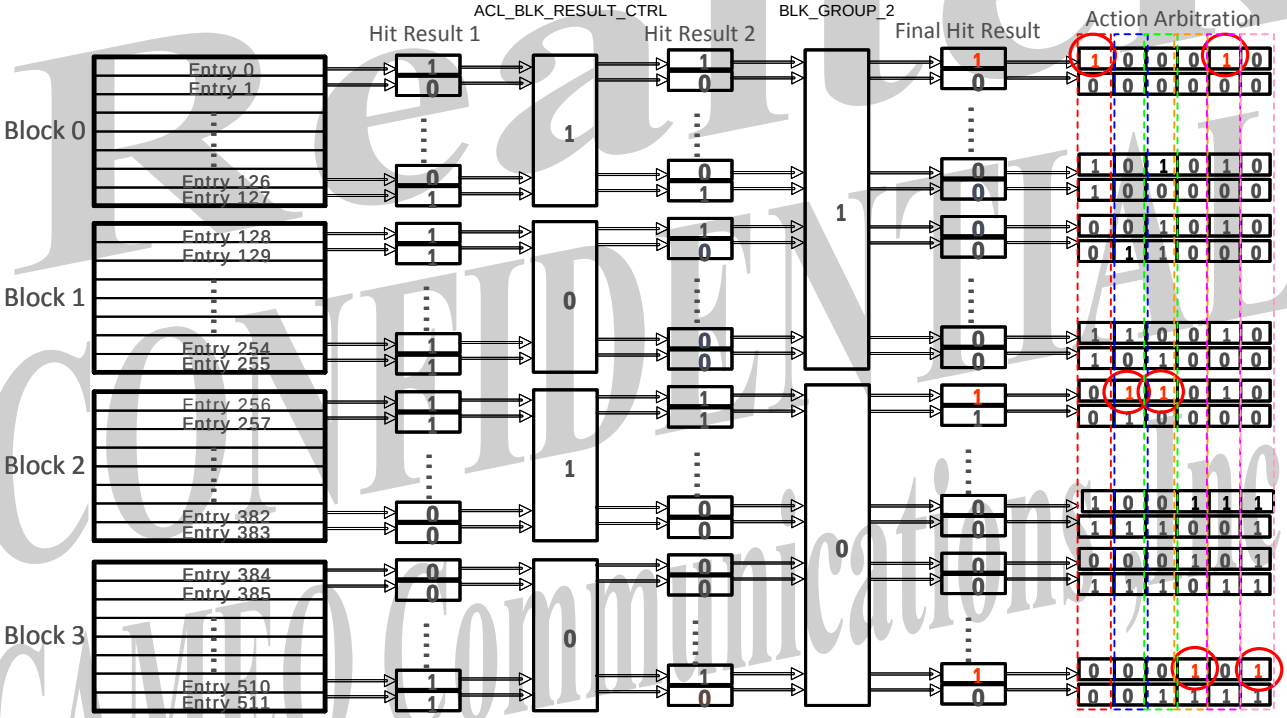


Figure 22: Action Arbitration and Execution

Statistic and Meter actions are the exceptions. The action arbitration and action execution is executed per block.

5.10 Clearance and Movement

The device supports the functionality to clear ACL entries to reduce CPU loading. Whole ACL entry includes ACL rule data, operations and actions are all cleared. The counters and meters indexed by the ACL entries will not be cleared.

Table 66: ACL_CLR_CTRL Register

Field Name	Bits	Description
CLR	1	Clear the ACL entries. Reset to 0 after clearing is completed.
CLR_TO	12	Clear to the index
CLR_FROM	12	Clear from the index

API REFERENCE

*int32 rtk_acl_rule_del(uint32 unit, rtk_acl_phase_t phase, rtk_acl_cleart_t *pClrIdx)*

ACL entry index represents priority because entry with lower index is hit first. Therefore, it is common to move entries to adjust the priority. The device supports the functionality to move ACL entries to reduce CPU loading. Whole ACL entry includes ACL rule data, operations and actions are all moved. The counters and meters indexed by the ACL entries will not be moved. Up bound movement is supported by setting MV_FROM to be larger than MV_TO while down bound movement is supported by setting MV_FROM to be lower than MV_TO.

Table 67: ACL_MV_CTRL Register

Field Name	Bits	Description
MV	1	Move the ACL entries. Reset to 0 after moving is completed.
MV_TO	12	Move to the index
MV_FROM	12	Move from the index

Table 68: ACL_MV_LEN_CTRL Register

Field Name	Bits	Description
MV_LEN	12	The number of entries to move

API REFERENCE

*int32 rtk_acl_rule_move(uint32 unit, rtk_acl_phase_t phase, rtk_acl_move_t *pData)*

5.11 Action Priority

- When the packet is hit statistic action, statistic action is always executed no matter the packet is drop/forward/trap by other modules.
- When the packet is hit mirror action, mirror action is executed in most cases except the packet is dropped by Ingress bandwidth control, Max frame size, Egress Port/Q congest, S-WRED modules.
- When the packet is hit both forward and meter actions, the meter action has higher priority.

5.12 Programming Example

Example 1: Copy ARP request to CPU and assign priority to ARP reply packet

Filter Condition:
ARP request is ARP packet and DMAC = broadcast MAC address.
ARP reply is ARP packet and DMAC = switch MAC address.

```
rtk_acl_templateIdx_t     template_info;
rtk_acl_igrAction_t       action ;

/* set cutline to 8 (0~7 is Ingress ACL and 8 ~15 is Egress ACL) */
rtk_acl_partition_set(unit, 8)

/* enable power and lookup function */
block_idx = 0;
rtk_acl_blockLookupEnable_set(unit, block_idx, ENABLED);
rtk_acl_blockPwrEnable_set(unit, block_idx, ENABLED);

/* use pre-defined template 0 */
template_info. template_id[0] = 0;
rtk_acl_templateSelector_set(unit, block_idx, template_info)

/** set entry 0 for ARP request packet **/
entry_id = 0

/* map entry to the first template */
data =0;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure frame type is ARP */
data = 0x0;
mask = 0x3;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FRAME_TYPE, data,
mask);

/* configure broadcast DMAC */
data = mask = 0xFFFFFFFFFFFF;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_DMAC, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
action.igr_acl.fwd_en = ENABLED;
action.igr_acl.fwd_data.fwd_type = ACL_IGR_ACTION_FWD_COPY_TO_PORTID;
action.igr_acl.fwd_data.fwd_info = CPU_PORT;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);

/* validate the entry */
rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);

/** set entry 1 for ARP reply packet **/
entry_id = 0

/* configure block template index */
data =0;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure frame type is ARP */
data = 0x0;
mask = 0x3;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FRAME_TYPE, data,
mask);
```

```
/* configure DMAC is switch MAC */
data = mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_SWITCHMAC, data,
mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
action.igr_acl.pri_en = ENABLED;
action.egr_acl.pri_data.pri = ARP_REQ_PRIO ;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);

/* validate the entry */
rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);
```

Example 2: IPv4 control filter and MAC control filter

MAC Filter condition: SMAC, DMAC, Ether Type, outer VID, and ingress port.
For ingress port 2, grant packet with DMAC = 11:22:33:44:55:66 && ether type = 0x0800 && outer VID = 1, and drop packet with DMAC = 11:22:33:44:55:XX.

Entry 1: DMAC = 11:22:33:44:55:66, ether type = 0x0800, OVID = 1, ingress port = 2.
Entry 2: DMAC = 11:22:33:44:55:XX, ingress port = 2.

IPv4 Filter condition: IPv4, SIP, DIP, IP protocol, L4 source/destination port, ICMP type/code, IGMP type, L4 src/dest port range, TOS/DSCP, and ingress port.

For ingress port 3 and 4, allow packet with
1. SIP = 10.1.1.0 through 10.1.1.255 && protocol = TCP && L4 src port less than 100
2. DIP = 20.1.1.1 && protocol = ICMP && ICMP type = 1 && ICMP code = 0
3. SIP = 10.1.1.1 && protocol = UDP && L4 dest port = 300 && DSCP = 40
4. DIP = 224.x.x.x && protocol = IGMP && IGMP type = 1

Otherwise, drop the IPv4 packet.

```
#define ACL_BLOCK_MAX_ENTRY 128

rtk_acl_templateIdx_t                templateIdx_info;
rtk_acl_template_t                    template_info
rtk_acl_rangeCheck_portMask_t    port_range;
rtk_acl_rangeCheck_l4Port_t       l4port_range;

/** block configuration **/
/* the filter entry will be configured in block 1 */
block_idx = 1;
rtk_acl_blockLookupEnable_set(unit, block_idx, ENABLED);
rtk_acl_blockPwrEnable_set(unit, block_idx, ENABLED);

/* use fixed template id 0 and configurable template id 5 */
/* template id 0 is for MAC control filter */
/* configure template id 5 for IP control filter */
mac_block_template_idx = 0;
ip_block_template_idx = 1;
mac_template_id = 0;
ip_template_id = 5;

template_info.field[0] = TMPLTE_FIELD_SIP0;
template_info.field[1] = TMPLTE_FIELD_SIP1;
template_info.field[2] = TMPLTE_FIELD_DIP0;
```

```
template_info.field[3] = TMPLTE_FIELD_DIP1;
template_info.field[4] = TMPLTE_FIELD_IP_TOS_PROTO;
template_info.field[5] = TMPLTE_FIELD_L4_SPORT;
template_info.field[6] = TMPLTE_FIELD_L4_DPORT;
template_info.field[7] = TMPLTE_FIELD_ICMP_IGMP;
template_info.field[8] = TMPLTE_FIELD_IP_L4PORT_RANG;
template_info.field[9] = TMPLTE_FIELD_SPMMASK;

rtk_acl_template_set(unit, ip_template_id, &template_info);

templateIdx_info. template_id[mac_block_template_idx] = mac_template_id;
templateIdx_info. template_id[ip_block_template_idx] = ip_template_id;
rtk_acl_templateSelector_set(unit, block_idx, templateIdx_info)


/** MAC control entry 1 **/
/* ACL configuration */
mac_entry1_prio = 0;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + mac_entry1_prio;

/* configure entry block template index */
data = mac_block_template_idx;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure DMAC */
data = 0x112233445566;
mask = 0xFFFFFFFFFFFF;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_DMAC, data, mask);

/* configure ether type */
data = 0x0800;
mask = 0xffff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_ETHERTYPE, data,
mask);

/* configure outer VID */
data = 0x1;
mask = 0xfff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_OTAG_VID, data,
mask);

/* configure ingress port */
data = 0x2;
mask = 0x3f;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_SPN, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
action.igr_acl.fwd_en = ENABLED;
action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_PERMIT;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);


rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);
```

```
/** MAC control entry 2 **/
/* ACL configuration */
mac_entry1_prio = 1;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + mac_entry1_prio;

/* configure entry block template index */
data = mac_block_template_idx;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure DMAC */
data = 0x112233445566;
mask = 0xFFFFFFFFFF00;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_DMAC, data, mask);

/* configure ingress port */
data = 0x2;
mask = 0x3f;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_SPN, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
action.igr_acl.fwd_en = ENABLED;
action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_DROP;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);


rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);


/** IP control entry 1 **/
/* ACL configuration */
ip_entry1_prio = 2;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + ip_entry1_prio;

/* configure entry block template index */
data = ip_block_template_idx;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure IPv4 frame type */
data = 0x2
mask = 0x3;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FRAME_TYPE, data,
mask);

/* configure SIP */
data = 0xA010100;
mask = 0xFFFFFF00;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP4_SIP, data, mask);

/* configure protocol */
data = 0x6;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP4PROTO_IP6NH,
data, mask);
```

```
/* configure L4 port range */
l4port_range_id = 0;

l4port_range.l4port_dir = RNGCHK_L4PORT_DIRECTION_SRC;
l4port_range. lower_bound = 0;
l4port_range. upper_bound = 99;
rtk_acl_rangeCheckL4Port_set(unit, l4port_range_id, &l4port_range);

data = mask = (1 << l4port_range_id);
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_PORT_RANGE, data,
mask);

/* configure ingress ports */
range_srcPort_id = 2;

LOGIC_PORTMASK_SET_PORT(port_range.portmask, 3) ;
LOGIC_PORTMASK_SET_PORT(port_range.portmask, 4) ;
rtk_acl_rangeCheckSrcPort_set(unit, range_srcPort_id, &port_range);

data = mask = (1 << range_srcPort_id);
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_SPMM, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
action.igr_acl.fwd_en = ENABLED;
action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_PERMIT;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);


rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);


/** IP control entry 2 **/
/* ACL configuration */
ip_entry2_prio = 3;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + ip_entry2_prio;

/* configure entry block template index */
data = ip_block_template_idx;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure IPv4 frame type */
data = 0x2
mask = 0x3;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FRAME_TYPE, data,
mask);

/* configure DIP */
data = 0x14010101;
mask = 0xFFFFFFFF;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP4_DIP, data, mask);

/* configure protocol */
data = 0x1;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP4PROTO_IP6NH,
data, mask);
```

```
/* configure ICMP type */
icmp_type = 1;

data = icmp_type;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_ICMP_TYPE, data,
mask);

/* configure ICMP code */
icmp_code = 0;

data = icmp_code;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_ICMP_CODE, data,
mask);

/* configure ingress ports */
range_srcPort_id = 2;
data = mask = (1 << range_srcPort_id);
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_SPMM, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
action.igr_acl.fwd_en = ENABLED;
action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_PERMIT;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);


rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);



/** IP control entry 3 **/
/* ACL configuration */
ip_entry3_prio = 4;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + ip_entry3_prio;

/* configure entry block template index */
data = ip_block_template_idx;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure IPv4 frame type */
data = 0x2
mask = 0x3;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FRAME_TYPE, data,
mask);

/* configure SIP */
data = 0xA010101;
mask = 0xFFFFFFFF;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP4_SIP, data, mask);

/* configure protocol */
data = 0x11;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP4PROTO_IP6NH,
```

```
data, mask);

/* configure L4 dest port */
l4dest_port = 300 ;
data = l4dest_port;
mask = 0xffff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_L4_DST_PORT, data,
mask);

/* configure DSCP */
data = 40;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP_DSCP, data, mask);

/* configure ingress ports */
range_srcPort_id = 2;
data = mask = (1 << range_srcPort_id);
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_SPMM, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
action.igr_acl.fwd_en = ENABLED;
action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_PERMIT;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);


rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);


/** IP control entry 4 **/
/* ACL configuration */
ip_entry4_prio = 5;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + ip_entry4_prio;

/* configure entry block template index */
data = ip_block_template_idx;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure IPv4 frame type */
data = 0x2
mask = 0x3;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FRAME_TYPE, data,
mask);

/* configure DIP */
data = 0xE0000000;
mask = 0xFF000000;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP4_DIP, data, mask);

/* configure protocol */
data = 0x2;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP4PROTO_IP6NH,
data, mask);

/* configure IGMP type */
igmp_type = 1;
```

```
data = igmp_type;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IGMP_TYPE, data,
mask);

/* configure ingress ports */
range_srcPort_id = 2;
data = mask = (1 << range_srcPort_id);
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_SPMM, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
action.igr_acl.fwd_en = ENABLED;
action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_PERMIT;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);


rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);



/** drop other IPv4 packet **/
/* ACL configuration */
ip_entry5_prio = 6;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + ip_entry5_prio;

/* configure IPv4 frame type */
data = 0x2
mask = 0x3;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FRAME_TYPE, data,
mask);

/* configure ingress ports */
range_srcPort_id = 2;
data = mask = (1 << range_srcPort_id);;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_SPMM, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
action.igr_acl.fwd_en = ENABLED;
action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_DROP;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);


rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);
```

Example 3: IPv6 control filter

IPv6 Filter condition: IPv6, SIPv6, DIPv6, next header, L4 source/destination port, ICMPv6 type/code, L4 src/dest port range, TOS/DSCP, and ingress port.

For ingress port 3 and 4, allow packet with

1. SIPv6 = EA03:0102::1 through EA03:0102::FFFF && DIPv6 = ED03:00A0::1234:3 && protocol = TCP && L4 src port great than 100
2. DIPv6 = link local && protocol = ICMPv6 && ICMPv6 type = 1 && ICMPv6 code = 0
3. SIPv6 = EA03:0102::1 && protocol = UDP && L4 dest port = 300 && DSCP = 40

Use Aggregation_1 operation to filter full SIPv6 and DIPv6.

```
#define ACL_BLOCK_MAX_ENTRY 128

rtk_acl_templateIdx_t                templateIdx_info;
rtk_acl_template_t                    template_info;
rtk_acl_operation_t                   oper;
rtk_acl_rangeCheck_portMask_t     port_range;
rtk_acl_rangeCheck_l4Port_t         l4port_range;

/** block configuration **/
/* the filter entry will be configured in block 2 */
block_idx = 2;
rtk_acl_blockLookupEnable_set(unit, block_idx, ENABLED);
rtk_acl_blockPwrEnable_set(unit, block_idx, ENABLED);

/* use fixed template id 3 and configurable template id 6 */
/* DIPv6, L4 src/dest port, ICMP type/code, next header, traffic class in fixed templated 3 */
/* SIPv6, L4 src/dest port range, ingress port mask in configurable template 6 */

/* configure template id 6 */
template_id = 6;

template_info.field[0] = TMPLTE_FIELD_SIP0;
template_info.field[1] = TMPLTE_FIELD_SIP1;
template_info.field[2] = TMPLTE_FIELD_SIP2;
template_info.field[3] = TMPLTE_FIELD_SIP3;
template_info.field[4] = TMPLTE_FIELD_SIP4;
template_info.field[5] = TMPLTE_FIELD_SIP5;
template_info.field[6] = TMPLTE_FIELD_SIP6;
template_info.field[7] = TMPLTE_FIELD_SIP7;
template_info.field[8] = TMPLTE_FIELD_IP_L4PORT_RANG;
template_info.field[9] = TMPLTE_FIELD_SPMMASK;

rtk_acl_template_set(unit, template_id, &template_info);

templateIdx_info. template_id[0] = 3;
templateIdx_info. template_id[1] = template_id;
rtk_acl_templateSelector_set(unit, block_idx, templateIdx_info)


/** IPv6 control entry 1 **/
/* ACL configuration */
ip6_entry1_prio = 0;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + (ip6_entry1_prio * 2);

/* configure entry block template index */
data = 0x0;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure IPv6 frame type */
data = 0x3
mask = 0x3;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FRAME_TYPE, data,
mask);
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, (entry_id + 1), USER_FIELD_FRAME_TYPE,
data, mask);
```

```
/* configure DIPv6 */
data = 0xED0300A000000000000000012340003;
mask = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP6_DIP, data, mask);

/* configure protocol */
data = 0x6;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP4PROTO_IP6NH,
data, mask);


/* configure entry block template index */
data = 0x1;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, (entry_id + 1), USER_FIELD_TEMPLATE_ID,
data, mask);

/* configure SIPv6 */
data = 0xEA030102000000000000000000000000;
mask = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, (entry_id + 1), USER_FIELD_IP6_SIP, data,
mask);

/* configure L4 port range */
l4port_range_id = 1;

l4port_range.l4port_dir = RNGCHK_L4PORT_DIRECTION_SRC;
l4port_range. lower_bound = 101;
l4port_range. upper_bound = 0xFFFF;
rtk_acl_rangeCheckL4Port_set(unit, l4port_range_id, &l4port_range);

data = mask = (1 << l4port_range_id);
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, (entry_id + 1), USER_FIELD_PORT_RANGE,
data, mask);

/* configure ingress ports */
range_srcPort_id = 2;

LOGIC_PORTMASK_SET_PORT(port_range.portmask, 3) ;
LOGIC_PORTMASK_SET_PORT(port_range.portmask, 4) ;
rtk_acl_rangeCheckSrcPort_set(unit, range_srcPort_id, &port_range);

data = mask = (1 << range_srcPort_id);
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, (entry_id + 1), USER_FIELD_SPMM, data,
mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
action.igr_acl.fwd_en = ENABLED;
action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_PERMIT;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);

/* configure AND1 operation */
oper.aggr_2 = ENABLED;
rtk_acl_ruleOperation_set(unit, ACL_PHASE_IGR_ACL, entry_id, &oper);

rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);
```

```
rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, (entryl_id + 1), ENABLED);



/** IPv6 control entry 2 **/
/* ACL configuration */
ip6_entry2_prio = 1;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + (ip6_entry2_prio * 2);

/* configure entry block template index */
data = 0;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure IPv6 frame type */
data = 0x3
mask = 0x3;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FRAME_TYPE, data,
mask);
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, (entry_id + 1), USER_FIELD_FRAME_TYPE,
data, mask);

/* configure DIPv6 */
data = 0xFE800000000000000000000000000000;
mask = 0xFFFF0000000000000000000000000000;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP6_DIP, data, mask);

/* configure protocol */
data = 0x1;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP4PROTO_IP6NH,
data, mask);

/* configure ICMP type */
icmp_type = 1;

data = icmp_type;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_ICMP_TYPE, data,
mask);

/* configure ICMP code */
icmp_code = 0;

data = icmp_code;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_ICMP_CODE, data,
mask);

/* configure entry block template index */
data = 1;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure ingress ports */
range_srcPort_id = 2;
data = mask = (1 << range_srcPort_id);
```

```
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_SPMM, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
action.igr_acl.fwd_en = ENABLED;
action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_PERMIT;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);

/* configure AND1 operation */
oper.aggr_2 = ENABLED;
rtk_acl_ruleOperation_set(unit, ACL_PHASE_IGR_ACL, (entry_id / 2), &oper);


rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);
rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, (entryl_id + 1), ENABLED);


/** IP control entry 3 **/
/* ACL configuration */
ip6_entry3_prio = 2;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + (ip6_entry3_prio * 2);

/* configure block template index */
data =0;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure IPv6 frame type */
data = 0x3;
mask = 0x3;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FRAME_TYPE, data,
mask);
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, (entry_id + 1), USER_FIELD_FRAME_TYPE,
data, mask);

/* configure protocol */
data = 0x11;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP4PROTO_IP6NH,
data, mask);

/* configure DSCP=40 */
data = 40;
mask = 0xff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_IP_DSCP, data, mask);

/* configure L4 dest port */
l4dest_port = 300 ;
data = l4dest_port;
mask = 0xffff;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_L4_DST_PORT, data,
mask);

/* configure block template index */
data =1;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, (entry_id + 1), USER_FIELD_TEMPLATE_ID,
data, mask);
```

```
/* configure SIP */
data = 0xEA030102000000000000000000000000;
mask = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, (entry_id + 1), USER_FIELD_IP6_SIP, data,
mask);

/* configure ingress ports */
range_srcPort_id = 2;
data = mask = (1 << range_srcPort_id);
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, (entry_id + 1), USER_FIELD_SPMM, data,
mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
action.igr_acl.fwd_en = ENABLED;
action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_PERMIT;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);

/* configure AND1 operation */
oper.aggr_2 = ENABLED;
rtk_acl_ruleOperation_set(unit, ACL_PHASE_IGR_ACL, (entry_id / 2), &oper);


rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);
rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, (entryl_id + 1), ENABLED);
```

Example 4: User define control filter

- Start 12 byte of packet is 0x1122
- Start 5 byte of L3 packet is 0x3344
- Start 3 byte of IPv4 header is 0x5566
- Start 0 byte of IP payload is 0x7788
- Start 9 byte of L4 payload is 0x99AA

```
#define ACL_BLOCK_MAX_ENTRY 128

rtk_acl_templateIdx_t                templateIdx_info;
rtk_acl_template_t                    template_info;
rtk_acl_fieldSelector_data_t        fs;

/** block configuration **/
/* the filter entry will be configured in block 3 */
block_idx = 3;
rtk_acl_blockLookupEnable_set(unit, block_idx, ENABLED);
rtk_acl_blockPwrEnable_set(unit, block_idx, ENABLED);

/* use configurable template id 7 */
template_id = 7;

template_info.field[0] = TMPLTE_FIELD_FIELD_SELECTOR_0;
template_info.field[1] = TMPLTE_FIELD_FIELD_SELECTOR_1;
template_info.field[2] = TMPLTE_FIELD_FIELD_SELECTOR_2;
template_info.field[3] = TMPLTE_FIELD_FIELD_SELECTOR_3;
template_info.field[4] = TMPLTE_FIELD_FIELD_SELECTOR_4;
template_info.field[5] = TMPLTE_FIELD_FIELD_SELECTOR_5;
```

```
template_info.field[6] = TMPLTE_FIELD_FIELD_SELECTOR_6;
template_info.field[7] = TMPLTE_FIELD_FIELD_SELECTOR_7;
template_info.field[8] = TMPLTE_FIELD_FIELD_SELECTOR_8;
template_info.field[9] = TMPLTE_FIELD_FIELD_SELECTOR_9;
template_info.field[10] = TMPLTE_FIELD_FIELD_SELECTOR_10;
template_info.field[11] = TMPLTE_FIELD_FIELD_SELECTOR_11;

rtk_acl_template_set(unit, template_id, &template_info);

templateIdx_info. template_id[0] = template_id;
rtk_acl_templateSelector_set(unit, block_idx, templateIdx_info)


/** user define control entry 1 **/
/* ACL configuration */
ud_entry1_prio = 0;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + ud_entry1_prio;

/* configure entry block template index */
data = 0x0;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure offset info */
fs.start = FS_START_POS_RAW;
fs.offset = 12;
rtk_acl_fieldSelector_set(unit, 0, &fs);

/* configure offset filter */
data = 0x1122;
mask = 0xFFFF;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FIELD_SELECTOR0,
data, mask);


/* configure offset info */
fs.start = FS_START_POS_L3;
fs.offset = 5;
rtk_acl_fieldSelector_set(unit, 1, &fs);

/* configure offset filter */
data = 0x3344;
mask = 0xFFFF;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FIELD_SELECTOR1,
data, mask);


/* configure offset info */
fs.start = FS_START_POS_IPV4;
fs.offset = 3;
rtk_acl_fieldSelector_set(unit, 2, &fs);

/* configure offset filter */
data = 0x5566;
mask = 0xFFFF;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FIELD_SELECTOR2,
data, mask);
```

```
/* configure offset info */
fs.start = FS_START_POS_IP;
fs.offset = 0;
rtk_acl_fieldSelector_set(unit, 3, &fs);

/* configure offset filter */
data = 0x7788;
mask = 0xFFFF;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FIELD_SELECTOR3,
data, mask);


/* configure offset info */
fs.start = FS_START_POS_L4;
fs.offset = 9;
rtk_acl_fieldSelector_set(unit, 4, &fs);

/* configure offset filter */
data = 0x99AA;
mask = 0xFFFF;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_FIELD_SELECTOR4,
data, mask);



rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);
```

6 Unicast Routing

The device supports IPv4 and IPv6 unicast routing by ACL. Network route and host route are supported by qualifying network address and host address respectively. Longest prefix match routing can also be archived when the ACL entries are arranged properly.

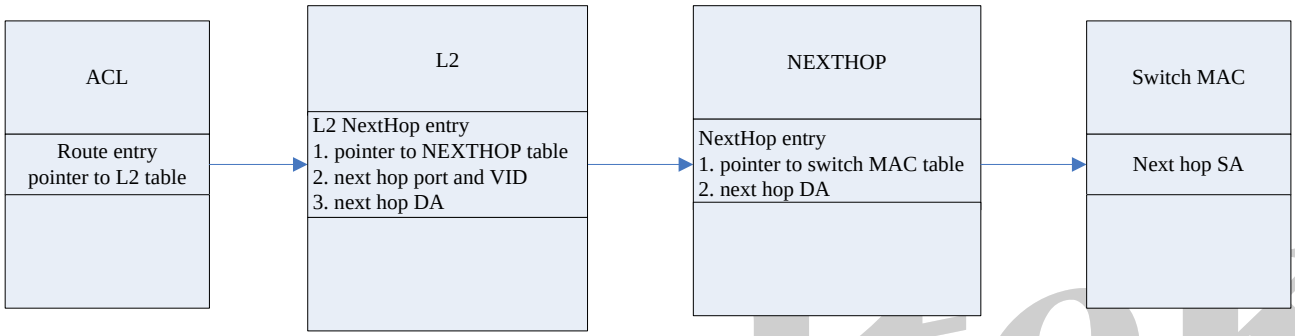


Figure 23: Routing Block Diagram

6.1 ACL Classification

Configure ACL rule data to match specific host/network address for host/network route. For the longest prefix match, the longer prefix entry should be inserted to lower ACL entry index for higher priority. In the situation, the default route entry is in the highest ACL entry index and the host route entry is in the lowest ACL entry index.

In addition to specify the rule data, configures ACL action to be routing and L2_NEXT_HOP index for pointing to L2 Next Hop entry in L2 table. If the pointed L2 entry is not a L2 Next Hop entry, the packet is dropped.

API REFERENCE

```
rtk_acl_ruleEntryField_read((uint32 unit, rtk_acl_phase_t phase, rtk_acl_id_t entry_idx,
      rtk_acl_fieldType_t type, uint8 *pData, uint8 *pMask);
rtk_acl_ruleEntryField_write((uint32 unit, rtk_acl_phase_t phase, rtk_acl_id_t entry_idx,
      rtk_acl_fieldType_t type, uint8 *pData, uint8 *pMask);

rtk_acl_ruleAction_get(uint32 unit, rtk_acl_phase_t phase, rtk_acl_id_t entry_idx,
      rtk_acl_action_t *pAction);
rtk_acl_ruleAction_set(uint32 unit, rtk_acl_phase_t phase, rtk_acl_id_t entry_idx,
      rtk_acl_action_t *pAction);
```

6.2 L2 Next Hop Entry

Type	Table Content (width = 76bits)										
Unicast	IP4 (1)	IP6 (1)	MAC (48)	SLP (6)	Age (3)	SABLK (1)	DABLK (1)	Static (1)	Suspend (1)	Next Hop (1)	NTO1_VID (12)

Type	Table Content (width = 76bits)											
NextHop	IP4 (1)	IP6 (1)	MAC (48)	DPN (6)	Age (3)	SABLK (1)	DABLK (1)	Static (1)	Suspend (1)	Next Hop (1)	VID_SEL (1)	NEXT_HOP_IDX (11)

Figure 24: L2 Unicast and NextHop Entry Format

The L2 next hop entry is a normal L2 unicast entry with next hop bit set by CPU. It is usually auto-learnt when receiving ARP packet from next hop and CPU should then set the next hop bit and NEXT_HOP_IDX before it is pointed by ACL entry.

Next hop port (DPN), VID (reverse calculated from L2 hash key) and NEXT_HOP_IDX are retrieved from L2 next hop entry for routing a packet. For the next hop VID, L2 next hop entry supports a bit VID_SEL to specify replacing inner or outer VLAN. And the NEXT_HOP_IDX is then used to lookup the NEXTHOP table for retrieving the next hop DMAC and switch MAC.

When port moving is occurred to a L2 next hop entry, the device updates the port information as usual. The L2 next hop entry can also be aged out, ROUTING_AGE action is triggered if a routing packet hits an aged out L2 next hop entry.

API REFERENCE

```
rtk_l2_addr_get(uint32 unit, rtk_l2_ucastAddr_t *pL2_addr)
rtk_l2_addr_set(uint32 unit, rtk_l2_ucastAddr_t *pL2_addr)
rtk_l2_addr_add(uint32 unit, rtk_l2_ucastAddr_t *pL2_addr)
```

6.3 Next Hop Table

The device supports a 2048 entries next hop table. Each entry contains a next hop DMAC address and a switch MAC address index. The switch MAC address index is then used to retrieve the switch MAC address.

Table 69: NextHop Table

Field Name	Bits	Description
GATEWAY_MAC	48	Next hop DMAC address
SW_MAC_IDX	4	Switch MAC address index

API REFERENCE

```
rtk_l3_routeEntry_get(uint32 unit, uint32 index, rtk_l3_routeEntry_t *pEntry)
rtk_l3_routeEntry_set(uint32 unit, uint32 index, rtk_l3_routeEntry_t *pEntry)
```

6.4 Switch MAC Address

The device supports 16 switch MAC addresses for routing interface. Different egress VLAN interface can then have different switch MAC address.

Table 70: ROUTING_SA_CTRL Register

Field Name	Bits	Description
SA	48	Switch MAC address

API REFERENCE

*rtk_l3_routeSwitchMacAddr_get(uint32 unit, uint32 index, rtk_mac_t *pMac)*
*rtk_l3_routeSwitchMacAddr_set(uint32 unit, uint32 index, rtk_mac_t *pMac)*

6.5 Routing Exception

Routing exception is asserted only if the packet is going to be routed. The supported routing exceptions are listed below:

- IPv4 Header Error
  - The IP header length is less than 20
- IPv4 TTL Exceed
  - Routing packet with TTL <= 1
- IPv4 Header with Option
- IPv6 Header Error
  - Version is not 6
- IPv6 Hop Limit Exceed
  - Routing packet with Hop Limit <= 1
- IPv6 packet with Hop-by-Hop header
- Gateway MAC Error
  - For IP routing, packet's DMAC should destine to the switch MAC address. If a packet is going to be routed but its DMAC address is not destined to one of the 16 switch MAC addresses, the packet will not be routed and the exception is asserted.
- Next Hop Aging Out

Table 71: ROUTING_EXCPT_CTRL Register

Field Name	Bits	Description
ROUTING_AGE	2	Action to take if L2 next hop entry is aged out 2'b00: drop 2'b01: forward 2'b10: trap 2'b11: reserved
GW_MAC_ERR	2	Action to take if packet's DMAC address is not destined to switch MAC address 2'b00: drop 2'b01: downgrade to L2 forwarding 2'b10: trap 2'b11: reserved
IP6_HOPBY_HOP	2	Action to take if IPv6 packet with hop-by-hop header 2'b00: drop 2'b01: forward 2'b10: trap 2'b11: reserved
IP6_HL_EXCEED	2	Action to take if hop limit reaches zero after doing routing. 2'b00: drop 2'b01: forward 2'b10: trap 2'b11: reserved
IP6_HDR_ERR	2	Action to take if IPv6 header error. 2'b00: drop 2'b01: forward 2'b10: trap
IP4_OPT	2	Action to take if IPv4 packet with option. 2'b00: drop

		2'b01: forward 2'b10: trap 2'b11: reserved
IP4_TTL_EXCEED	2	Action to take if TTL reaches zero after doing routing. 2'b00: drop 2'b01: forward 2'b10: trap 2'b11: reserved
IP4_HDR_ERR	2	Action to take if IPv4 header error. 2'b00: drop 2'b01: forward 2'b10: trap 2'b11: reserved

API REFERENCE

Corresponding Routing Exception Type:
ROUTE_EXCEPTION_TYPE_HDR_ERR
ROUTE_EXCEPTION_TYPE_TTL_EXCEED
ROUTE_EXCEPTION_TYPE_WITH_OPT
ROUTE_EXCEPTION_TYPE_IP6_HDR_ERR
ROUTE_EXCEPTION_TYPE_IP6_HL_EXCEED
ROUTE_EXCEPTION_TYPE_IP6_HOP_BY_HOP
ROUTE_EXCEPTION_TYPE_GW_MAC_ERR
ROUTE_EXCEPTION_TYPE_ENTRY_AGE_OUT

*rtk_trap_routeExceptionAction_get(uint32 unit, rtk_trap_routeExceptionType_t type, rtk_action_t *pAction)*
rtk_trap_routeExceptionAction_set(uint32 unit, rtk_trap_routeExceptionType_t type, rtk_action_t action)

6.6 Programming Example

Example:

ACL entry priority: host route > network route > default route

L3 entry 1: DIP = 10.1.1.2/32, VID = 2, MAC = 00:12:34:56:78:91, port = 3
L3 entry 2: DIP = 10.1.2.0/24, VID = 3, MAC = 00:12:34:56:78:92, port = 4
L3 entry 3: DIP = 10.1.0.0/16, VID = 4, MAC = 00:12:34:56:78:93, port = 5
L3 entry 4: DIP = 0.0.0.0/0, VID = 5, MAC = 00:12:34:56:78:94, port = 6

VLAN 1: VID = 2, MAC = 00:11:22:33:44:51
VLAN 2: VID = 3, MAC = 00:11:22:33:44:52
VLAN 3: VID = 4, MAC = 00:11:22:33:44:53
VLAN 4: VID = 5, MAC = 00:11:22:33:44:54

```
#define ACL_BLOCK_MAX_ENTRY 128

rtk_l3_routeEntry_t    route_entry;
rtk_l2_ucastAddr_t    l2_entry;

/** VLAN configuration **/
/* set VLAN 1 switch MAC */
v1_switchMAC_idx = 0;
v1_mac[0] = 0x00;
```

```
v1_mac[0] = 0x11;
v1_mac[0] = 0x22;
v1_mac[0] = 0x33;
v1_mac[0] = 0x44;
v1_mac[0] = 0x51;
rtk_l3_routeSwitchMacAddr_set(unit, v1_switchMAC_idx, &v1_mac);

/* set VLAN 2 switch MAC */
v2_switchMAC_idx = 1;
v2_mac[0] = 0x00;
v2_mac[0] = 0x11;
v2_mac[0] = 0x22;
v2_mac[0] = 0x33;
v2_mac[0] = 0x44;
v2_mac[0] = 0x52;
rtk_l3_routeSwitchMacAddr_set(unit, v2_switchMAC_idx, &v2_mac);

/* set VLAN 3 switch MAC */
v3_switchMAC_idx = 2;
v3_mac[0] = 0x00;
v3_mac[0] = 0x11;
v3_mac[0] = 0x22;
v3_mac[0] = 0x33;
v3_mac[0] = 0x44;
v3_mac[0] = 0x53;
rtk_l3_routeSwitchMacAddr_set(unit, v3_switchMAC_idx, &v3_mac);

/* set VLAN 4 switch MAC */
v4_switchMAC_idx = 3;
v4_mac[0] = 0x00;
v4_mac[0] = 0x11;
v4_mac[0] = 0x22;
v4_mac[0] = 0x33;
v4_mac[0] = 0x44;
v4_mac[0] = 0x54;
rtk_l3_routeSwitchMacAddr_set(unit, v4_switchMAC_idx, &v4_mac);

/** Next Hop entry configuration **/
/* L3 entry1 next hop */
route_entry_idx = 0;
route_entry.swMac_idx = v1_switchMAC_idx;
route_entry.hostMac[0] = 0x00;
route_entry.hostMac[0] = 0x12;
route_entry.hostMac[0] = 0x34;
route_entry.hostMac[0] = 0x56;
route_entry.hostMac[0] = 0x78;
route_entry.hostMac[0] = 0x91;
rtk_l3_routeEntry_set(unit, route_entry_idx, &route_entry);

osal_memcpy(&l2_entry.mac, &route_entry.hostMac, ETHER_ADDR_LEN);
l2_entry.vid = 2;
l2_entry.flags |= RTK_L2_UCAST_FLAG_NEXTHOP;
l2_entry.route_idx = route_entry_idx;
rtk_l2_addr_add(unit, &l2_entry);
l2_nexthop_entry1 = l2_entry.l2_idx;

/* L3 entry2 next hop */
route_entry_idx = 1;
route_entry.swMac_idx = v2_switchMAC_idx;
```

```
route_entry.hostMac[0] = 0x00;
route_entry.hostMac[0] = 0x12;
route_entry.hostMac[0] = 0x34;
route_entry.hostMac[0] = 0x56;
route_entry.hostMac[0] = 0x78;
route_entry.hostMac[0] = 0x92;
rtk_l3_routeEntry_set(unit, route_entry_idx, &route_entry);

osal_memcpy(&l2_entry.mac, &route_entry.hostMac, ETHER_ADDR_LEN);
l2_entry.vid = 3;
l2_entry.flags |= RTK_L2_UCAST_FLAG_NEXTHOP;
l2_entry.route_idx = route_entry_idx;
rtk_l2_addr_add(unit, &l2_entry);
l2_nexthop_entry2 = l2_entry.l2_idx;

/* L3 entry3 next hop */
route_entry_idx = 2;
route_entry.swMac_idx = v3_switchMAC_idx;
route_entry.hostMac[0] = 0x00;
route_entry.hostMac[0] = 0x12;
route_entry.hostMac[0] = 0x34;
route_entry.hostMac[0] = 0x56;
route_entry.hostMac[0] = 0x78;
route_entry.hostMac[0] = 0x93;
rtk_l3_routeEntry_set(unit, route_entry_idx, &route_entry);

osal_memcpy(&l2_entry.mac, &route_entry.hostMac, ETHER_ADDR_LEN);
l2_entry.vid = 4;
l2_entry.flags |= RTK_L2_UCAST_FLAG_NEXTHOP;
l2_entry.route_idx = route_entry_idx;
rtk_l2_addr_add(unit, &l2_entry);
l2_nexthop_entry3 = l2_entry.l2_idx;

/* L3 entry4 next hop */
route_entry_idx = 3;
route_entry.swMac_idx = v3_switchMAC_idx;
route_entry.hostMac[0] = 0x00;
route_entry.hostMac[0] = 0x12;
route_entry.hostMac[0] = 0x34;
route_entry.hostMac[0] = 0x56;
route_entry.hostMac[0] = 0x78;
route_entry.hostMac[0] = 0x94;
rtk_l3_routeEntry_set(unit, route_entry_idx, &route_entry);

osal_memcpy(&l2_entry.mac, &route_entry.hostMac, ETHER_ADDR_LEN);
l2_entry.vid = 5;
l2_entry.flags |= RTK_L2_UCAST_FLAG_NEXTHOP;
l2_entry.route_idx = route_entry_idx;
rtk_l2_addr_add(unit, &l2_entry);
l2_nexthop_entry4 = l2_entry.l2_idx;

/** ACL configuration **/
/* block configuration */
block_idx = 0;
rtk_acl_blockLookupEnable_set(unit, block_idx, ENABLED);
rtk_acl_blockPwrEnable_set(unit, block_idx, ENABLED);

/* use fixed template id 1 */
route_block_template_idx = 1;
```

```
templateIdx_info. template_id[mac_block_template_idx] = route_template_id;
templateIdx_info. template_id[ip_block_template_idx] = route_template_id;
rtk_acl_templateSelector_set(unit, block_idx, templateIdx_info)

/** L3 entry 1 **/
entry1_prio = 0;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + entry1_prio;

/* configure entry block template index */
data = route_block_template_idx;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure DIP */
data = 0x0A010102;
mask = 0xFFFFFFFF;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_DMAC, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
acl_action.igr_acl.fwd_en = ENABLED;
acl_action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_UNICAST_ROUTING;
acl_action.igr_acl.fwd_data.fwd_info = l2_nexthop_entry1;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);

rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);

/** L3 entry 2 **/
entry2_prio = 1;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + entry2_prio;

/* configure entry block template index */
data = route_block_template_idx;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure DIP */
data = 0x0A010200;
mask = 0xFFFFFF00;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_DMAC, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
acl_action.igr_acl.fwd_en = ENABLED;
acl_action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_UNICAST_ROUTING;
acl_action.igr_acl.fwd_data.fwd_info = l2_nexthop_entry2;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);

rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);

/** L3 entry 3 **/
entry3_prio = 2;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + entry3_prio;

/* configure entry block template index */
data = route_block_template_idx;
```

```
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure DIP */
data = 0x0A010000;
mask = 0xFFFF0000;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_DMAC, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
acl_action.igr_acl.fwd_en = ENABLED;
acl_action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_UNICAST_ROUTING;
acl_action.igr_acl.fwd_data.fwd_info = l2_nexthop_entry3;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);

rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);

/** L3 entry 4 **/
entry4_prio = 3;
entry_id = (block_idx * ACL_BLOCK_MAX_ENTRY) + entry4_prio;

/* configure entry block template index */
data = route_block_template_idx;
mask = 0x1;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_TEMPLATE_ID, data,
mask);

/* configure DIP */
data = 0x00000000;
mask = 0x00000000;
rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, entry_id, USER_FIELD_DMAC, data, mask);

/* configure action */
osal_memset(&action, 0, sizeof(rtk_acl_igrAction_t));
acl_action.igr_acl.fwd_en = ENABLED;
acl_action.igr_acl.fwd_data.fwd_type = ACL_ACTION_FWD_UNICAST_ROUTING;
acl_action.igr_acl.fwd_data.fwd_info = l2_nexthop_entry4;
rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL , entry_id, &action);

rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, entryl_id, ENABLED);
```

7 Spanning Tree

Spanning tree is a Layer 2 link management protocol that provides path redundancy while preventing loops in the network. For a Layer 2 network to function properly, only one active path can exist between any two stations. The system provides a number of hardware features to support IEEE 802.1D STP, IEEE 802.1W RSTP, and IEEE 802.1s MSTP which includes per port per instance spanning tree port state configuration, the VLAN to instance mapping in 4K VLAN table, and so on.

7.1 Spanning Tree State Configuration

The device provides 256 instances of spanning tree port state configuration. Each instance contains port state for each port, and the state can be disabled(0), blocking/listening(1), learning(2), forwarding(3) which can map to STP/ RSTP/ MSTP states as below table.

STP State Mapping	RSTP State Mapping	MSTP State Mapping	STATE_PORT Configuration
Disabled	Disabled	Disabled	Disabled(0)
Blocking/Listening	Discard	Discard	Blocking/Listening(1)
Learning	Learning	Learning	Learning(2)
Forwarding	Forwarding	Forwarding	Forwarding(3)

The limitations of packet receive and transmit for each state is shown as below:

Port State	Receive Non-BPDU	Receive BPDU	Transmit Packet (without ignore-stp)	Transmit Packet (with ignore-stp)	Learn Address	Forward Frame
Disabled	No	No	No	No	No	No
Blocking/Listening	No	Yes	No	Yes	No	No
Learning	No	Yes	No	Yes	Yes	No
Forwarding	Yes	Yes	Yes	Yes	Yes	Yes

Note: ‘ignore-stp’ is a signal from CPU tag (from CPU_TX_TAG.BP_FLTR_2 bit). Please refer to CPU Tag chapter for the detail.

2’b00 – Disabled
- All packets including RMA/BPDU are dropped on ingress/egress port.
- MAC address is not learnt.

2’b01 – Blocking/Listening
- Only RMA/BPDU packets may/could be sent to the CPU, other packets are dropped.
- MAC address is not learnt.
- Packet without Ignore-stp can’t be transmitted.
- If RMA configuration specifies the particular packet to bypass the STP ingress checking, then the packet can pass the STP ingress filter. Detail please refers to RMA chapter.

2’b10 – Learning
- Only RMA/BPDU packets may/could be sent to the CPU, other packets are dropped after Learning Process.
- Packet without Ignore-stp can’t be transmitted.
- If RMA configuration specifies the particular packet to bypass the STP ingress checking, then the packet can pass the STP ingress filter. Detail please refers to RMA chapter.

2’b11 – Forwarding
- All packets are learnt and forwarded.

Table 72: MSTI Table

Field Name	Bits	Description
STATE_PORT0 - STATE_PORT51	1	STP Port State. 2'b00: Disabled 2'b01: Blocking/Listening 2'b10: Learning 2'b11: Forwarding

API REFERENCE

```
rtk_stp_mstpState_get(uint32 unit, uint32 msti, rtk_port_t port, rtk_stp_state_t *pStp_state);
rtk_stp_mstpState_set(uint32 unit, uint32 msti, rtk_port_t port, rtk_stp_state_t stp_state);
```

7.2 Spanning Tree Instance

The device supports 256 instances of spanning tree port states, and the instances are mapped by VLAN to support MSTP per port per VLAN configuration. The FID_MSTI field of VLAN entry in VLAN table indicates the mapping instance ID. For example, if FID_MSTI of VLAN 100 is configured value 1, the spanning tree port state of instance 1 is used to determine the packet learning, receiving, and forwarding for the packet received from VLAN 100.

For the VLAN unaware protocol STP and RSTP, only instance 0 (CIST) is needed and it applies to all VLANs. To reduce the software effort modifying FID_MSTI field of VLAN entries when system switching between VLAN aware (MSTP) and VLAN unaware (STP/RSTP) protocol, global register field MSTI_MODE is provided to determine the instance source. If MSTI_MODE is set to normal mode, the mapped instance ID is from FID_MSTI value which suits the behavior of MSTP; if MSTI_MODE is set to force mode, the mapped instance ID of all VLANs is always 0 which suits the behavior of STP and RSTP. By configuring MSTI_MODE, software doesn't need to modify FID_MSTI of 4k VLAN entries when changing spanning tree protocols between MSTP and STP/RSTP.

Table 73: ST_CTRL Register

Field Name	Bits	Description
MSTI_MODE	1	Indicate instance ID is either from FID_MSTI of VLAN table or force using 0 (CIST). 0b0: normal mode (from VLAN table) 0b1: force mode (force using 0) Note: The configuration only affects MSTI but not FID.

API REFERENCE

```
rtk_stp_mstpInstance_create(uint32 unit, uint32 msti);
rtk_stp_mstpInstance_destroy(uint32 unit, uint32 msti);
rtk_stp_isMstpInstanceExist_get(uint32 unit, uint32 msti, uint32 *pMsti_exist);
rtk_stp_mstpInstanceMode_get(uint32 unit, rtk_stp_mstiMode_t *pMsti_mode);
rtk_stp_mstpInstanceMode_set(uint32 unit, rtk_stp_mstiMode_t msti_mode);
```

8 Traffic Isolation

Typical applications for traffic isolation might be in a Co-location Facility or IaaS network (Infrastructure as a Service Cloud), where you might have several customers using the same subnet, but communications between the customers is not desirable as it would circumvent their firewalls. Another common use for traffic isolation might be in a hotel situation, where each hotel room has internet access, all are on the same subnet, but communications between the rooms is not desired.

Traffic isolation separates all the traffic (unicast, multicast, and broadcast) in layer 2 to prevent traffic interference and bandwidth utilization. The system provides two type isolation modules, Port-based Isolation and VLAN-based Isolation. The details are as below.

8.1 Port-based Isolation

When hosts communicate with router through switch ports, Port Isolation can be utilized to allow the communication between host and the router but communication between hosts are forbidden. Port Isolation is a function for an ingress port to uni-directional defines the ports it can communicate with. All the traffic regardless unicast, multicast and broadcast are limited by Port Isolation function. For each ingress port, the device provides a port isolation port mask register.

Table 74: PORT_ISO_CTRL Register

Field Name	Bits	Description
P_ISO_MBR_1	21	Specify ports (port 52 ~ port 32) to communicate.
P_ISO_MBR_0	32	Specify ports (port 31 ~ port 0) to communicate.

Figure 1 is an example that hosts A and B are connected to ports 0 and 3 respectively while router is connected to port 7. The traffic between A and B must be forwarded by router. Therefore, user can configure the 7th bit of Port Isolation port mask of ports 0 and 3 to be 1 while other bits should be configured to 0, and Port Isolation port mask of port 7 still remains all ports. Then, direct communication between downlink ports, 0 and 3, is forbidden and only traffic between uplink port and downlink port can be forwarded。

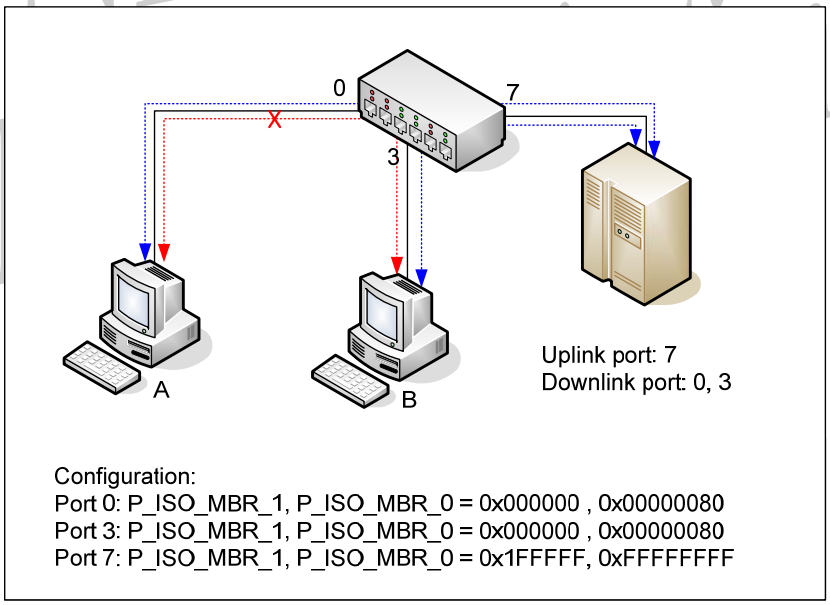


Figure 25: Example of Uplink Port and Downlink Port for Port Isolation

API REFERENCE

```
rtk_port_isolation_get(uint32 unit, rtk_port_t port, rtk_portmask_t *pPortmask);
rtk_port_isolation_set(uint32 unit, rtk_port_t port, rtk_portmask_t portmask);
rtk_port_isolation_add(uint32 unit, rtk_port_t port, rtk_port_t iso_port);
rtk_port_isolation_del(uint32 unit, rtk_port_t port, rtk_port_t iso_port);
```

8.2 VLAN-based Isolation

There is an application that uplink ports can communicate with uplink port and downlink ports, but downlink ports aren't allowed to communicate with each other in a certain VLAN domain. VLAN-based Isolation can be used to meet the application. The device provides 16-set VLAN-based Isolation configuration registers PORT_ISO_VB_ISO_PM_CTRL. For each configuration, it can specify a VLAN range by VID_LOWER and VID_UPPER registers and using VALID register to enable or disable the entry. Besides, system applies register PORT_ISO_VB_CTRL.VLAN_TYPE to select the VLAN source for VLAN range comparison. Take a look to below examples:

Example 1: Received packet with outer VID 100 and inner VID 200. VLAN_PORT_FWD defined in VLAN module is configured to Inner VLAN, and ingress VLAN translation is disabled.
When VLAN_TYPE = 0, Comparing VID = 200.
When VLAN_TYPE = 1, Comparing VID = 100.
When VLAN_TYPE = 2, Comparing VID = 200.

Example 2: Same as example 1, but ingress VLAN translation is enabled that outer VID 100 is translated to 150 and (inner VID 200 is translated to 250.
When VLAN_TYPE = 0, Comparing VID = 250.
When VLAN_TYPE = 1, Comparing VID = 150.
When VLAN_TYPE = 2, Comparing VID = 250.

Table 75: PORT_ISO_VB_CTRL Register

Field Name	Bits	Description
VLAN_TYPE	2	VLAN ID source for comparison. 2'b00: inner VLAN 2'b01: outer VLAN 2'b10: forwarding VLAN 2'b11: reserved

Table 76: PORT_ISO_VB_ISO_PM_CTRL Register

Field Name	Bits	Description
VB_ISO_MBR_1	21	VLAN-based isolated port mask (port 52 to 32). Ports configured as 1 in this port mask could communicate with all ports belonging to the same specified VLAN. Ports configured as 0 could only communicate with ports configured as 1 belonging to the same specified VLAN.
VB_ISO_MBR_0	32	VLAN-based isolated port mask (port 31 to 0). Ports configured as 1 in this port mask could communicate with all ports belonging to the same specified VLAN. Ports configured as 0 could only communicate with ports configured as 1 belonging to the same specified VLAN.
VID_UPPER	12	VLAN ID upper bound.
VID_LOWER	12	VLAN ID lower bound.
VALID	1	Validate the VLAN-based isolation entry. 1'b0: entry invalid 1'b1: entry valid

For VLAN-based isolation, a 53-bit port mask is supported to defined communication port list for each entry. The ports with bit value 0 can't communicate with other ports having same bit value 0. However, they can communicate with the other port with bit value 1. If bit value of ports is set to 1, the port can communicate to all ports in the VLAN. Usually, bit value of uplink port is set to 1 and bit value of downlink port is set to 0 in network environment. For example: ports 0 and 1 are uplink ports and other ports are downlink ports. By configuring (VB_ISO_MBR_1, VB_ISO_MBR_0) of VLAN-based Isolation entry to be (0x0, 0x3), then uplink ports can communicate to all ports, but downlink ports only communicate to uplink port.

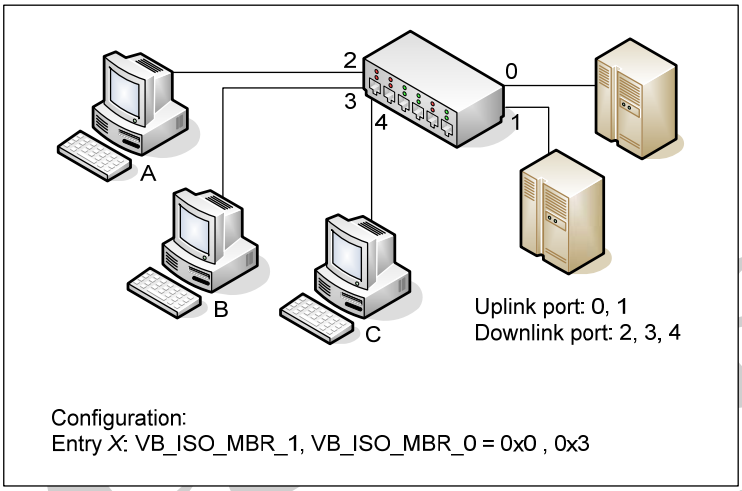


Figure 26: Example of Uplink Port and Downlink Port for VLAN-based Isolation

API REFERENCE

```
rtk_port_vlanBasedIsolationEntry_get(uint32 unit, uint32 index, rtk_port_vlanIsolationEntry_t* pEntry);
rtk_port_vlanBasedIsolationEntry_set(uint32 unit, uint32 index, rtk_port_vlanIsolationEntry_t* pEntry);
rtk_port_vlanBasedIsolation_vlanSource_get(uint32 unit, rtk_port_vlanIsolationSrc_t *pVlanSrc);
rtk_port_vlanBasedIsolation_vlanSource_set(uint32 unit, rtk_port_vlanIsolationSrc_t vlanSrc);
```

PROGRAMMING EXAMPLE

Adding VLAN-based Isolation Entry and VLAN-based Isolation VID Type Configuration

```
/* Configure VLAN-based isolation entry value */
rtk_port_vlanIsolationEntry_t entry;
entry.enable = ENABLED;
entry.vid = 1;
entry.vid_high = 100;
entry.portmask.bits[1] = 0x0;
entry.portmask.bits[0] = 0x3;

/* Add VLAN-based isolation entry */
rtk_port_vlanBasedIsolationEntry_set(unit, 0, &entry);


/* Configure inner VID to be VLAN-based isolation comparing VID */
vlanSrc = VLAN_ISOLATION_SRC_INNER;
rtk_port_vlanBasedIsolation_vlanSource_set(unit, vlanSrc) ;
```

9 Reserved Multicast Address (RMA)

RMA supported by the device is 01-80-C2-00-00-00 through 01-80-C2-00-00-2F. These MAC address is defined as standards group MAC by IEEE. The RMA configuration described in the document doesn't apply to PAUSE frame (01-80-C2-00-00-01).

9.1 Learning

Protocols, such as STP and LACP which use port MAC as the source MAC of the control frame, it is useless to learn the port MAC. Thus, the device per RMA can specify whether to learn the source MAC address of a RMA frame. Spanning tree port state has higher priority than configuration register RMA_SMAC_LRN_CTRL.

Table 77: RMA_SMAC_LRN_CTRL Register

Field Name	Bits	Description
LRN	1	Learning source MAC address for RMA packet. 1'b0: Not learn 1'b1: Learn

API REFERENCE

*rtk_trap_rmaLearningEnable_get(uint32 unit, rtk_mac_t *pRma_frame, rtk_enable_t *pEnable)*
*rtk_trap_rmaLearningEnable_set(uint32 unit, rtk_mac_t *pRma_frame, rtk_enable_t enable)*

9.2 RMA Action

The device supports Forward/Drop/Trap actions for each RMA. The Forward action is to lookup L2 table first. If lookup hit, the packet is forwarded to the portmask specified by the L2 multicast entry. Otherwise, the packet is flooded to the portmask defined in VLAN Profile.
Because RMA is a kind of L2 multicast traffic, it is also applied the L2 multicast lookup miss action in addition to RMA action by default. The forwarding behavior for L2 multicast and RMA may need to be different and independent, for example, drop L2 unknown multicast traffic but forward RMA(BPDU) packet. Thus, the device supports a configuration register RMA_CTRL_3 to specify whether to apply L2 multicast lookup miss action on RMA packet.

Table 78: RMA_CTRL_3 Register

Field Name	Bits	Description
BYPASS_LM_ACT_EN	1	Apply L2 multicast lookup miss action for RMA frame. 1'b0: Disable 1'b1: Enable

API REFERENCE

*rtk_trap_rmaAction_get(uint32 unit, rtk_mac_t *pRma_frame, rtk_trap_rma_action_t *pRma_action)*
*rtk_trap_rmaAction_set(uint32 unit, rtk_mac_t *pRma_frame, rtk_trap_rma_action_t rma_action)*
*rtk_trap_rmaLookupMissActionEnable_get(uint32 unit, rtk_enable_t *pEnable)*
rtk_trap_rmaLookupMissActionEnable_set(uint32 unit, rtk_enable_t enable)

9.3 BPDU

RMA action is per RMA basis except BPDU, PTP and LLDP. For BPDU (01-80-C2-00-00-00), the device supports per port configuration register to define the forwarding behavior.

Table 79: RMA_PORT_BPDU_CTRL Register

Field Name	Bits	Description
ACT	2	BPDU forwarding action. 2'b00: Forward 2'b01: Drop 2'b10: Trap to CPU 2'b11: Flood

There is an application that a BPDU received from a STP disabled port needs forwarding to other STP disabled ports regardless VLAN. Therefore, in addition to support BPDU per port forwarding action, the device further supports an extra action "Flood" for BPDU (01-80-C2-00-00-00). The Flood action is similar to Forward action except it bypasses the egress VLAN filtering. Unlike other RMAs that flooding portmask is retrieved from VLAN Profile, the BPDU flooding portmask is specified by RMA_BPDU_FLD_PMSK register which can specify to all STP disabled port in the application.

Table 80: RMA_BPDU_FLD_PMSK Register

Field Name	Bits	Description
PMSK	53	Flooding portmask for BPDU when BPDU action is Flood.

API REFERENCE

*rtk_trap_portMgmtFrameAction_get(uint32 unit, rtk_port_t port, rtk_trap_mgmtType_t frameType, rtk_action_t *pAction)*
rtk_trap_portMgmtFrameAction_set(uint32 unit, rtk_port_t port, rtk_trap_mgmtType_t frameType, rtk_action_t action)
*rtk_trap_bpduFloodPortmask_get(uint32 unit, rtk_portmask_t *pflood_portmask)*
*rtk_trap_bpduFloodPortmask_set(uint32 unit, rtk_portmask_t *pflood_portmask)*

9.4 PTP and LLDP

The MAC address of PTP and LLDP is 01-80-C2-00-00-0E. The EtherType of PTP is 0x88F7 while LLDP is 0x88CC. The device supports per port configuration register to define the forwarding behavior for PTP and LLDP.

Table 81: RMA_PORT_PTP_CTRL Register

Field Name	Bits	Description
ACT	2	PTP forwarding action. 2'b00: Forward 2'b01: Drop 2'b10: Trap to CPU 2'b11: Reserved

Table 82: RMA_PORT_LLDP_CTRL Register

Field Name	Bits	Description
ACT	2	LLDP forwarding action. 2'b00: Forward

	2'b01: Drop 2'b10: Trap to CPU 2'b11: Flood (across VLAN and flooding portmask is retrieved from VLAN Profile)

Whether to learn the MAC address of PTP and LLDP is controlled by PTP_LRN and LLDP_LRN register fields.

Table 83: RMA_MGN_LRN_CTRL Register

Field Name	Bits	Description
PTP_LRN	1	PTP source MAC address learning. 1'b0: Not learn 1'b1: Learn
LLDP_LRN	1	LLDP source MAC address learning. 1'b0: Not learn 1'b1: Learn

API REFERENCE

*rtk_trap_portMgmtFrameAction_get(uint32 unit, rtk_port_t port, rtk_trap_mgmtType_t frameType, rtk_action_t *pAction)*
rtk_trap_portMgmtFrameAction_set(uint32 unit, rtk_port_t port, rtk_trap_mgmtType_t frameType, rtk_action_t action)
*rtk_trap_mgmtFrameLearningEnable_get(uint32 unit, rtk_trap_mgmtType_t frameType, rtk_enable_t *pEnable)*
rtk_trap_mgmtFrameLearningEnable_set(uint32 unit, rtk_trap_mgmtType_t frameType, rtk_enable_t enable)

9.5 STP Ingress Filtering

The device supports to bypass STP ingress port state filtering. The bypass action works only if the RMA action is specified to "Trap to CPU". If STP ingress filtering is bypassed, the source MAC address learning is decided by RMA_SMAC_LRN_CTRL register and STP port state.

Table 84: RMA_CTRL_3 Register

Field Name	Bits	Description
PTP_BYPASS_STP	1	Bypass STP Ingress Filtering for PTP. 1'b0: Disable 1'b1: Enable (only works if RMA_PORT_PTP_CTRL.ACT is Trap to CPU)
LLDP_BYPASS_STP	1	Bypass STP Ingress Filtering for LLDP. 1'b0: Disable 1'b1: Enable (only works if RMA_PORT_LLDP_CTRL.ACT is Trap to CPU)
RMA_0X_BYPASS_STP	1	Bypass STP Ingress Filtering for RMA 01-80-C2-00-00-02 through 01-80-C2-00-00-0F exclude 01-80-C2-00-00-02. 1'b0: Disable 1'b1: Enable (only works if RMA_XX_ACT is Trap to CPU)
SLOW_PROTO_BYPASS_STP	1	Bypass STP Ingress Filtering for slow protocol (01-80-C2-00-00-02). 1'b0: Disable 1'b1: Enable (only works if RMA_02_ACT is Trap to CPU)

API REFERENCE

*rtk_trap_bypassStp_get(uint32 unit, rtk_trap_bypassStpType_t frameType, rtk_enable_t *pEnable)*
rtk_trap_bypassStp_set(uint32 unit, rtk_trap_bypassStpType_t frameType, rtk_enable_t enable)

9.6 VLAN Ingress Filtering

The device supports to bypass VLAN ingress filtering. The bypass action works only if the RMA action is specified to "Trap to CPU". If VLAN ingress filtering is bypassed, the source MAC address learning is decided by RMA_SMAC_LRN_CTRL register and STP port state.

Table 85: RMA_CTRL_3 Register

Field Name	Bits	Description
PTP_BYPASS_VLAN	1	Bypass VLAN ingress filtering for PTP. 1'b0: Disable 1'b1: Enable (only works if RMA_PORT_PTP_CTRL.ACT is Forward or Trap to CPU)
LLDP_BYPASS_VLAN	1	Bypass VLAN ingress filtering for LLDP. 1'b0: Disable 1'b1: Enable (only works if RMA_PORT_LLDP_CTRL.ACT is Forward or Trap to CPU)
RMA_00_BYPASS_VLAN	1	Bypass VLAN ingress filtering for RMA 01-80-C2-00-00-00. 1'b0: Disable 1'b1: Enable (only works if RMA_00_ACT is Forward or Trap to CPU)
RMA_02_BYPASS_VLAN	1	Bypass VLAN ingress filtering for RMA 01-80-C2-00-00-02. 1'b0: Disable 1'b1: Enable (only works if RMA_02_ACT is Forward or Trap to CPU)
RMA_0E_BYPASS_VLAN	1	Bypass VLAN ingress filtering for RMA 01-80-C2-00-00-0E. 1'b0: Disable 1'b1: Enable (only works if RMA_0E_ACT is Forward or Trap to CPU)
RMA_0X_BYPASS_VLAN	1	Bypass VLAN ingress filtering for RMA 01-80-C2-00-00-01 through 01-80-C2-00-00-0F exclude 01-80-C2-00-00-02 and 01-80-C2-00-00-0E. 1'b0: Disable 1'b1: Enable (only works if RMA_0X_ACT is Forward or Trap to CPU)

API REFERENCE

*rtk_trap_bypassVlan_get(uint32 unit, rtk_trap_bypassVlanType_t frameType, rtk_enable_t *pEnable)*
rtk_trap_bypassVlan_set(uint32 unit, rtk_trap_bypassVlanType_t frameType, rtk_enable_t enable)

9.7 User Defined RMA

The device supports two user defined RMA. The user define RMA is to configure specific MAC address as RMA. The behavior of user defined RMA is same as general RMA.

Table 86: RMA_USR_DEF_CTRL_SET Register

Field Name	Bits	Description
ADDR	48	Specific MAC address
BYPASS_VLAN	1	Bypass VLAN ingress filtering. 1'b0: Disable 1'b1: Enable (only works if RMA_USR_DEF_CTRL_SETX_1.ACT is

		Forward or Trap to CPU)
LRN	1	Learning behavior. 1'b0: Not learn 1'b1: Learn
BYPASS_STP	1	Bypass STP ingress filtering. 1'b0: Disable 1'b1: Enable (only works if RMA_USR_DEF_CTRL_SETX_1.ACT is Trap to CPU)
ACT	2	Forwarding action. 2'b00: Forward 2'b01: Drop 2'b10: Trap to CPU 2'b11: Flood (across VLAN and flooding portmask is retrieved from VLAN Profile)
EN	1	Enable user defined RMA. 1'b0: Disable 1'b1: Enable

The RMA configurations may have confliction due to specify same MAC address. The priority sequence is user defined RMA 0 > user defined RMA 1 > BPDU/PTP/LLDP/Other RMA.

API REFERENCE

*rtk_trap_userDefineRma_get(uint32 unit, uint32 userDefine_idx, rtk_trap_userDefinedRma_t *pUserDefinedRma)*
*rtk_trap_userDefineRma_set(uint32 unit, uint32 userDefine_idx, rtk_trap_userDefinedRma_t *pUserDefinedRma)*

*rtk_trap_bypassVlan_get(uint32 unit, rtk_trap_bypassVlanType_t frameType, rtk_enable_t *pEnable)*
rtk_trap_bypassVlan_set(uint32 unit, rtk_trap_bypassVlanType_t frameType, rtk_enable_t enable)

*rtk_trap_userDefineRmaLearningEnable_get(uint32 unit, uint32 userDefine_idx, rtk_enable_t *pEnable)*
rtk_trap_userDefineRmaLearningEnable_set(uint32 unit, uint32 userDefine_idx, rtk_enable_t enable)

*rtk_trap_bypassStp_get(uint32 unit, rtk_trap_bypassStpType_t frameType, rtk_enable_t *pEnable)*
rtk_trap_bypassStp_set(uint32 unit, rtk_trap_bypassStpType_t frameType, rtk_enable_t enable)

*rtk_trap_userDefineRmaAction_get(uint32 unit, uint32 userDefine_idx, rtk_trap_rma_action_t *pAction)*
rtk_trap_userDefineRmaAction_set(uint32 unit, uint32 userDefine_idx, rtk_trap_rma_action_t action)

*rtk_trap_userDefineRmaEnable_get(uint32 unit, uint32 userDefine_idx, rtk_enable_t *pEnable)*
rtk_trap_userDefineRmaEnable_set(uint32 unit, uint32 userDefine_idx, rtk_enable_t enable)

10 Link Aggregation

Trunk is a mechanism to aggregate physical ports to a logical port for higher bandwidth. The device provides 16 trunk group configurations. Each trunk group can aggregate at most 8 ports. For trunk ports traffic balancing, a distribution function is supported with configurable distribution parameters. There are four sets of distribution parameter configurations, and each trunk group can bind to a set of configuration. The device also provides traffic separation mechanism for isolating known multicast traffic and flooding traffic.

10.1 Trunk Member

TRK_MBR register field specifies the trunk member ports. At most 8 ports can be configured as member port. Only the first 8 member ports are taken if member ports are configured more than 8 ports.

API REFERENCE

```
rtk_trunk_port_get(uint32 unit, uint32 trk_gid, rtk_portmask_t *pTrunk_member_portmask);
rtk_trunk_port_set(uint32 unit, uint32 trk_gid, rtk_portmask_t *pTrunk_member_portmask);
```

10.2 Load Balancing

Traffic to a trunk port is re-distributed to a trunk member port by distribution function. Which distribution parameter (packet characteristic) should participate the distribution function is determined by a 7-bit configuration register field HASH_MSK. The supported distribution parameters are listed below:

- SPA(bit 0): source physical port
- SMAC(bit 1): source MAC address
- DMAC(bit 2): destination MAC address
- SIP(bit 3): source IP
- DIP(bit 4): destination IP
- SPORT(bit 5): source layer 4 port
- DPORT(bit 6): destination layer 4 port

The distribution parameter can be any combination of above. For example, HASH_MSK=0x7 means taking SPA, SMAC, DMAC as distribution parameters. The device supports four distribution parameter configurations, and each trunk group can bind to one of them.

The distribution function does XOR operation for every 5 bits of distribution parameters and produces a 5 bits distribution result. Each distribution parameter can be shifted several bits before XOR operation to avoid traffic unbalanced because of the way of folding the distribution parameters. For example, if the distribution parameter SPA shifts 2 bits, the key folding is:

Figure 27: Distribution Parameter Shifting Example

The register fields to configure shift bits are SPA_SHIFT_BITS, SMAC_SHIFT_BITS, DMAC_SHIFT_BITS,

SIP_SHIFT_BITS, DIP_SHIFT_BITS, SPORT_SHIFT_BITS, and DPORT_SHIFT_BITS. Each distribution parameter configuration supports dedicated shift configuration registers.

The 5 bits distribution result (range from 0 ~ 31) is divided by the number of candidate member ports, and the remainder decides the packet's egress port. The candidate ports are link up member ports that are not dedicated for separated traffic (separated traffic please refer to next section). Egress port is (remainder+1)th link member port counting from least ID. For example, if distribution result is 10 and there are three candidate trunk member ports 1,3,5. The remainder is 1 and thus 2nd candidate port "port 3" will be the egress port.

API REFERENCE

```
rtk_trunk_distributionAlgorithmBind_get(uint32 unit, uint32 trk_gid, uint32 *pAlgo_idx);
rtk_trunk_distributionAlgorithmBind_set(uint32 unit, uint32 trk_gid, uint32 algo_idx);
rtk_trunk_distributionAlgorithmParam_get(uint32 unit, uint32 algo_idx, uint32 *pAlgo_bitmask);
rtk_trunk_distributionAlgorithmParam_set(uint32 unit, uint32 algo_idx, uint32 algo_bitmask);
rtk_trunk_distributionAlgorithmShift_get(uint32 unit, uint32 algo_idx, rtk_trunk_distAlgoShift_t *pShift);
rtk_trunk_distributionAlgorithmShift_set(uint32 unit, uint32 algo_idx, rtk_trunk_distAlgoShift_t *pShift);
```

10.3 Traffic Separation

Traffic is forwarded to trunk member port accouding to distribution function, thus the available bandwidth for the traffic is not stationary. Thus, a streaming traffic may be flow controlled because it is forwared to an egress port that is also used by a low priority high speed traffic. Another application is to separate flodding traffic from other traffic since flooding traffic has more chance to be malevolence. The device provides a traffic separation mechanism to separate known multicast traffic or flooding(L2 lookup miss) traffic to a specific port. By configuring SEPARATE_TRAFFIC register field (per trunk group configuration), the device could disable traffic separation, separate known multicast traffic, separate flooding traffic, or separate both known multicast traffic and flooding traffic.

The link up member port with largest port ID is the dedicated port to forward separated traffic. The port would not be a distribution candidate port if traffic separation is enable. For example, if link up trunk member ports are port 1,3,5,7, and traffic separation is configured to separate known multicast traffic, port 7 would be dedicated to forward known multicast traffic, and other traffic would forward through port 1,3,5.

Traffic separation mechanism would automatically stop if there is 0 or only 1 link up trunk member port. It works only when there is more than 1 link up member ports.

Table 87: TRK_SEP_TRAFFIC_CTRL Register

Field Name	Bits	Description
SEP_TRAFFIC	1	Separate traffic for a trunk group. 2'b00: disable 2'b01: separate known multicast traffic 2'b10: separate floodng traffic 2'b11: separate both known multicast and flooding traffic

API REFERENCE

```
rtk_trunk_trafficSeparate_get(uint32 unit, uint32 trk_gid, rtk_trunk_separateType_t *pSeparateType);
rtk_trunk_trafficSeparate_set(uint32 unit, uint32 trk_gid, rtk_trunk_separateType_t separateType);
```

11 Mirror

Mirror is a popular mechanism for diagnostic. By configuring interest patern of traffic (such as Ingress/Egress port, or classification rules), a copy of mirrored traffic would be forwarded to mirroring port for further analysis. The device supports four mirroring set configurations. Each mirroring set can have a mirroring port and multiple mirrored ports. The mirror criteria can be specified by ingress portmask, egress portmask, or flow based (by ACL). To mirror traffic across switches, RSPAN is also supported.

Since mirror is a traffic monitor feature, the mirroring traffic should not trigger flow control to affect traffic forwording of mirrored ports. The device would automatically stop mirroring if the mirroring port buffer reaches flow control threshold. Mirroring would be automatically back to work when the mirroring port has resource to forward mirrored traffic.

11.1 Ingress/Egress Mirror

Ingress mirror is to mirror the orignal ingress packets while egress mirror is to mirror the modified egress packets. For example, a VLAN untag packet received from port 1 which forwarded to port 2 with VLAN tagged. If port 1 is ingress mirrored, a copy of the packet without VLAN tag would be sent to mirroring port. If port 2 is egress mirrored, a copy with VLAN tag would be sent to mirroring port. For ingress mirror, both good and bad packets such as CRC error, under-size, over-size, alignment error packets would be mirrored. For egress mirror, only packets not dropped by the device can be mirrored. For example, the packets dropped by egress bandwidth control would not be egress mirrored.

The device supports four mirroring set. Enable status of each mirroring set is decided by MIR_EN register field and MIR_DST_P is used to specify the mirroring port. Each mirroring set can configure SPM_0 and SPM_1 to specify the ports for doing ingress mirror. For example, to mirror packets received from port 1-4 should configure SPM_0 to 0xf and SPM_1 to 0x0, and set MIR_EN to 1 for a mirroring set.

Egress mirror portmask is specified by DPM_0 and DPM_1. If any outgoing ports of packets hit the portmask, the packet would be egress mirrorred. For example, configure DPM_0 to 0x10 (port 5) and DPM_1 to 0x0, and a multicast packet outgoing to port 1,3,5 would be mirrored since it has a egress port which is port 5.

If a user wants to specify both ingress portmask and egress portmask for mirror condition. MIR_OP is used to define the AND/OR operation between SPM and DPM. If MIR_OP is OR, the packet either comes from SPM or forwards to DPM would be mirrored. If MIR_OP is AND, only the packet comes from SPM and forwads to DPM would be mirrored. Thus, if SPM or DPM is all zero, the mirror is never satisfied.

For a packet hit both SPM and DPM that configued in the same mirroring set, the device follows MIR_SEL to do either ingress or egress mirror and only a copy of mirrored packet would be sent to mirroring port. If MIR_SEL is 0, the egress modified packet would be mirrored; if MIR_SEL is 1, the original ingress packet would be mirrored.

DST_P_ISO can be used to isolate mirroring port from receiving normal forwarded traffic. If a packet is both normal forwarded and mirrored to mirroring port, two packets would be sent to mirroring port: one is normal forwarded, another is mirrored.

Table 88: Mirror Entry Related Registers

Register Name	Field Name	Bits	Description
MIR_CTRL	MIR_EN	1	Enable the mirroring set. 1'b0: disable 1'b1: enable
	MIR_SEL	1	Specify which direction of packet would be mirrored if both ingress/egress mirror are hit. 1'b0: egress

			1'b1: ingress
	MIR_OP	1	Specify the AND/OR operation between SPM and DPM. 1'b0: OR 1'b1: AND
	DST_P_ISO	1	Isolate the mirroring port from normal forwarding traffic. 1'b0: The mirroring port accepts both mirrored traffic and normal forwarding traffic. 1'b1: The mirroring port only accepts mirrored traffic.
	MIR_DST_P	6	Mirroring port.
MIR_SPM_CTRL	SPM_0	32	Bitmap to select ingress mirrored port (port 0 to 31). 1'b0: disable 1'b1: enable
	SPM_1	21	Bitmap to select ingress mirrored port (port 32 to 52). 1'b0: disable 1'b1: enable
MIR_DPM_CTRL	DPM_0	32	Bitmap to select egress mirrored port (port 0 to 31). 1'b0: disable 1'b1: enable
	DPM_1	21	Bitmap to select egress mirrored port (port 32 to 52). 1'b0: disable 1'b1: enable

API REFERENCE

```
rtk_mirror_group_get(uint32 unit, uint32 mirror_id,rtk_mirror_entry_t *pMirrorEntry);
rtk_mirror_group_set(uint32 unit,uint32 mirror_id, rtk_mirror_entry_t *pMirrorEntry);
```

11.2 Flow Based Mirror

A specific flow such as mirroring IPv4 traffic only can be done by flow based mirror. Flow based mirror is done by Ingress or Egress ACL with mirror action. ACL mirror action can index a mirror entry (MIRROR_INDEX), and the packet which qualified by ACL would be mirrored to the mirroring port specified by the indexed mirror entry. Please refer to Ingress ACL mirror action and Egress ACL mirror action for the detail.

If a packet hit multiple ACL mirror rules, only the rule with lowest ID would take effect. For example, if ACL 100 mirror SA=00:00:00:00:00:01 packet to mirror entry 3; ACL 200 mirror DA=ff:ff:ff:ff:ff:ff packet to mirror entry 2; ACL 300 mirror IPv4 packet to mirror entry 1. If a packet hits all the packet charisterics, it would be mirrored to mirroring port of entry 3 since ACL 100 has lowest ID. However, a packet could mirrorred by both Ingress and Egress ACL concurrently if the indexed mirror entries are different.

Table 89: Flow Based Mirror Related ACL action Fields

Field Name	Bits	Description
MIRROR_INDEX	2	Mirror set index Note: Ingress and Egress ACL have this field
MIRROR_ACT	1	Mirror original or modified packet 0: Original 1: Modified Note: Only Egress ACL has this field.

11.3 RSPAN

Remote switched port analysis (RSPAN) mechanism is to mirror traffic cross network. Switches in RSPAN architecture could play roles as a source switch, an intermediate switch, or a destination switch.

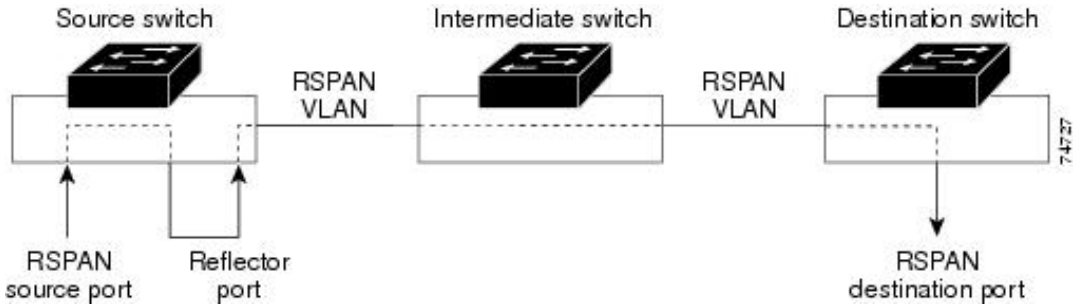


Figure 28: RSPAN Topology Illustration

The source switch collects mirrored traffic (behaves like local mirror), and send a copy of mirrored traffic to reflector port. An RSPAN tag would be added to mirrored traffic when it sends out from reflector port. The system provides per mirroring set RSPAN_TX_TAG_ADD register field to control whether to add RSPAN tag for the mirroring traffic. The TPID, Priority, CFI, VID fields of RSPAN tag are controlled by per mirroring set register RSPAN_TAG_TPID_IDX, RSPAN_TAG_PRI, RSPAN_TAG_CFI, and RSPAN_TAG_VID. The RSPAN_TAG_TPID_IDX is the index to VLAN outer tag TPID pool, and the detail of configuring VLAN outer tag TPID pool please refer to VLAN chapter.

The RSPAN traffic forwarding of intermedia switch is simply based on VLAN forwarding. Since RSPAN traffic path may be conflict with real data path, it should not be learnt. VLAN Profile configuration can be utilized to disable L2 MAC learning for RSPAN VLAN.

The process of RSPAN destination switch is to identify RSPAN tag, forward it to destination port, and remove the RSPAN tag. To support RSPAN destination switch, register field RSPAN_RX_TAG_EN and RSPAN_RX_TAG_RM must be enabled for a mirroring set. With RSPAN_RX_TAG_EN enabled, the packets received from any ports of the switch that match the TPID that indexed by RSPAN_TAG_TPID_IDX and RSPAN_TAG_VID would be mirrored to mirroring port of the mirroring set. It is suggested to keep DPM and SPM of the mirroring set empty, or the outgoing traffic of mirroring port would contain local mirrored traffic and RSPAN traffic. With RSPAN_RX_TAG_RM enabled, the RSPAN tag of RSPAN traffic would be removed when it outgoes to mirroring port so traffic analyzer could recevie the original mirrored data.

Table 90: Mirror Entry Related Registers

Register Name	Field Name	Bits	Description
MIR_RSPAN_TX_CTRL	RSPAN_TX_TAG_ADD	1	Add RSPAN VLAN tag for mirrored packets that do not carry RSPAN VLAN tag for mirroring set n. This bit is ignored by packets which already have RSPAN VLAN tag. 0b0: disable 0b1: enable
MIR_RSPAN_RX_TAG_RM_CTRL	RSPAN_RX_TAG_RM	1	If RSPAN_RX_TAG_EN is enabled, this bit decides if the RSPAN VLAN tag of RSPAN packet should be removed. This bit is ignored by packets that do not have RSPAN VLAN tag. 0b0: disable 0b1: enable
MIR_RSPAN_RX_TAG_EN_CTRL	RSPAN_RX_TAG_EN	1	Enable the ingress RSPAN VLAN tag identification ability. If it is identified as RSPAN traffic, it would be mirrored to MIR_DST_P. 0b0: disable 0b1: enable
MIR_RSPAN_VLAN_CTRL	RSPAN_TAG_TPID_IDX	2	Index to VLAN OTPID pool. This field decides the TPID of RSPAN tag. This is used to identify ingress RSPAN traffic and to decide the TPID of egress added RSPAN VLAN tag.

	RSPAN_TAG_PRI	3	Priority of RSPAN tag. This is used to decide PRI of the egress added RSPAN VLAN tag.
	RSPAN_TAG_CFI	1	CFI of RSPAN tag. This is used to decide CFI of the egress added RSPAN VLAN tag.
	RSPAN_TAG_VID	12	VID of RSPAN tag. This is used to identify ingress RSPAN traffic and to decide VID of the egress added RSPAN VLAN tag.

API REFERENCE

*rtk_mirror_rspanIgrMode_get(uint32 unit, uint32 mirror_id, rtk_mirror_rspanIgrMode_t *pIgrMode);*
rtk_mirror_rspanIgrMode_set(uint32 unit, uint32 mirror_id, rtk_mirror_rspanIgrMode_t igrMode);
*rtk_mirror_rspanEgrMode_get(uint32 unit, uint32 mirror_id, rtk_mirror_rspanEgrMode_t *pEgrMode)*
rtk_mirror_rspanEgrMode_set(uint32 unit, uint32 mirror_id, rtk_mirror_rspanEgrMode_t egrMode)
*rtk_mirror_rspanTag_get(uint32 unit, uint32 mirror_id, rtk_mirror_rspanTag_t *pTag)*
*rtk_mirror_rspanTag_set(uint32 unit, uint32 mirror_id, rtk_mirror_rspanTag_t *pTag)*

12 sFlow Sampling

Packet sampling is typically performed by the ASICs, provides continuously monitor traffic flow at wire speed on all ports simultaneously. The sampled packet is trapped to CPU. CPU then encapsulates it to sFlow datagram and sends to a central server for further analyzing.

12.1 Port-based sampling

sFlow is a packet-based sampling technology embedded in switches and routers for monitoring network. Base on per port configured sampling rate (N), an average of 1 out of N packets is randomly sampled. The sampling rate (N) is per port per direction configurable and set through IGR_RATE and EGR_RATE fields.

The sampled packet is sent to CPU with CPU tag and marked the sampling reason in SFLOW field of CPU tag. The SFLOW field in CPU tag represents port number of sFlow egress sampling. It represents sFlow ingress sampling if its value is 0x3E and the ingress port can be retrieved from CPU tag too. The field is nonsense if its value is 0x3F.

If a packet is sampled at multiple egress ports concurrently, only the packet transmitted from the least port is sampled.

Table 91: SFLOW_PORT_RATE_CTRL Register

Field Name	Bits	Description
EGR_RATE	16	Packet sampling rate for egress traffic. The value is 0 represent the function is disabled.
IGR_RATE	16	Packet sampling rate for ingress traffic. The value is 0 represent the function is disabled.

API REFERENCE

```
rtk_mirror_sflowPortEgrSampleRate_get(uint32 unit, rtk_port_t port, uint32 *pRate);
rtk_mirror_sflowPortEgrSampleRate_set(uint32 unit, rtk_port_t port, uint32 rate);

rtk_mirror_sflowPortIgrSampleRate_get(uint32 unit, rtk_port_t port, uint32 *pRate);
rtk_mirror_sflowPortIgrSampleRate_set(uint32 unit, rtk_port_t port, uint32 rate);
```

12.2 sFlow compensation

Suppose the packet has got remarked for sampling ingress and egress. In that case, SMPL_SEL field becomes the deciding factor, which one has to be sampled. The next packet which satisfies the other direction will be compensated.

Table 92: SFLOW_CTRL Register

Field Name	Bits	Description
SMPL_SEL	1	If any packet is marked for sampling twice for both Ingress & Egress then this bit will dictate which packet will be sampled, either Ingress packet or Egress packet. 0b0: Ingress Sampling 0b1: Egress Sampling

API REFERENCE

```
rtk_mirror_sflowSampleCtrl_get(uint32 unit, rtk_sflowSampleCtrl_t *pCtrl);
rtk_mirror_sflowSampleCtrl_set(uint32 unit, rtk_sflowSampleCtrl_t ctrl);
```

12.3 Flow-based sampling

The flow-based sampling is constructed from ACL and mirror configuration. Configure the ACL entry to match to specific flow then bind with the mirror entry.

The mirror sampling rate is per mirroring set configuration. It is used to support packet sampling mechanism. If the mirror entry's sampling rate is set to N. The device skips every (N – 1) packets and samples one packet to mirror entry's destination port.

Table 93: Mirror Sample Related Register

Register Name	Field Name	Bits	Description
MIR_SAMPLE_RATE_CTRL	SAMPLE_RATE	16	Packet sampling rate of mirrored packet. 0x0: just mirror, do not need to care about sampling rate 0xn: sample the mirrored packets every n packets
MIR_CTRL	MIR_DST_P	6	Mirroring port

API REFERENCE

```
rtk_mirror_sflowMirrorSampleRate_get(uint32 unit, uint32 mirror_id, uint32 *pRate);
rtk_mirror_sflowMirrorSampleRate_set(uint32 unit, uint32 mirror_id, uint32 rate);
```

PROGRAMMING EXAMPLE

Sample packet with SMAC = 0001:0203:0405 to CPU and mirror entry's sample rate is 10.

```
/* ACL configuration */
acl_id = 5;

/* configure ACL rule qualify */
data = 0x000102030405;
mask = 0xFFFFFFFFFFFF;

rtk_acl_ruleEntryField_write(unit, ACL_PHASE_IGR_ACL, acl_id, USER_FIELD_SMAC , data, mask);

/* configure ACL rule action */
mirror_id = 1;

acl_action.igr_acl.mirror_en = ENABLED;
acl_action.igr_acl.mirror_data.mirror_set_idx = mirror_id;

rtk_acl_ruleAction_set(unit, ACL_PHASE_IGR_ACL, acl_id, &acl_action);

rtk_acl_ruleValidate_set(unit, ACL_PHASE_IGR_ACL, acl_id, ENABLED);

/* mirror configuration */
mirror_entry.mirror_enable = ENABLED;
mirror_entry. mirroring_port = 52;
```

```
rate = 10;

rtk_mirror_group_set(unit, mirror_id, &mirror_entry);
rtk_mirror_sflowMirrorSampleRate_set(unit, mirror_id, rate);
```

13 Ingress Bandwidth Control

To limit ingress port traffic rate, system provides ingress bandwidth control function for each ingress port (exclude CPU port). When ingress bandwidth exceeds the relevant configured-bandwidth, system could be configured to drop packet or pause the sender. Ingress bandwidth control function could be configured to admit some special traffic type, such as BPDU, RMA, IGMP, ARP and Realtek control packet (Ethertype = 0x8899).

13.1 Bandwidth Configuration

For each ingress port, the device provides the register IGR_BWCTRL_PORT_CTRL.PORT_EN in Table 94 and IGR_BWCTRL_PORT_CTRL_10G.PORT_EN in Table 95 configurations to enable or disable the ingress bandwidth control function. For Giga port, the ingress bandwidth range is 16Kbps~1Gbps configured by register IGR_BWCTRL_PORT_CTRL.RATE. For 10G port, the configured range is 16Kbps~10Gbps configured by register IGR_BWCTRL_PORT_CTRL_10G.RATE. The granularity for ingress bandwidth control is 16Kbps.

Table 94: IGR_BWCTRL_PORT_CTRL Register

Field Name	Bits	Description
HI_THR	16	High threshold of ingress bandwidth control of the port. Unit: 128 Bytes
RATE	16	Ingress bandwidth control rate of the port. Unit: 16 Kbps Control Rate = IGR_BWCTRL_PORT_CTRL.RATE * 16 Kbps
FC_EN	1	Enable flow control of the port. FC_EN takes effect only when the port enables ingress bandwidth control function in register PORT_EN. 1'b0: disable. Drop packet. 1'b1: enable. Send PAUSE ON/OFF frame to sender.
PORT_EN	1	Enable ingress bandwidth control function of the port 1'b0: disable 1'b1: enable

Table 95: IGR_BWCTRL_PORT_CTRL_10G Register

Field Name	Bits	Description
HI_THR	16	High threshold of ingress bandwidth control of the port. Unit: 128 Bytes
RATE	20	Ingress bandwidth control rate of the port. Unit: 16 Kbps Control Rate = IGR_BWCTRL_PORT_CTRL. RATE * 16 Kbps
FC_EN	1	Enable flow control of the port. FC_EN takes effect only when the port enables ingress bandwidth control function in register PORT_EN. 1'b0: disable. Drop packet. 1'b1: enable. Send PAUSE ON/OFF frame to sender.
PORT_EN	1	Enable ingress bandwidth control function of the port 1'b0: disable 1'b1: enable

Table 96: IGR_BWCTRL_CTRL_LB_THR Register

Field Name	Bits	Description
LOW_THR	16	Low threshold of ingress bandwidth control of the port. Unit: 128 Bytes

Since ingress bandwidth control function is enabled, the excess bandwidth packet would be dropped or paused

according to the per-port configuration IGR_BWCTRL_PORT_CTRL.FC_EN in Table 94 and IGR_BWCTRL_PORT_CTRL_10G.FC_EN in Table 95. The device provided global low-threshold defined in IGR_BWCTRL_CTRL_LB_THR.LOW_THR register in Table 96 and per-port high-threshold defined in IGR_BWCTRL_PORT_CTRL.HI_THR and IGR_BWCTRL_PORT_CTRL_10G.HI_THR as Figure . The high threshold is used to trigger sending PAUSE ON packet or dropping packet in traffic over high-threshold. System sends PAUSE OFF packet or stops dropping packet until traffic is lower than the low threshold.

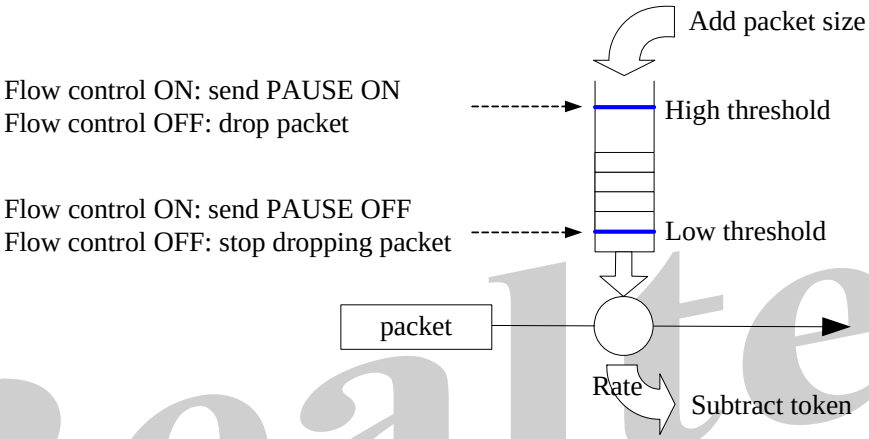


Figure 29: Ingress Bandwidth Control Leaky Bucket Diagram

For overflow monitoring, system provides per-port register IGR_BWCTRL_PORT_EXCEED_FLG.EXCEED_FLG to log the exceed status whether the traffic ever beyond the high-threshold on the port. This could be useful for some security application.

Table 97: IGR_BWCTRL_PORT_EXCEED_FLG Register

Field Name	Bits	Description
EXCEED_FLG	1	Exceed status of the port. Write 1 to clear. 1'b0: not exceed 1'b1: exceed

API REFERENCE

```
rtk_rate_igrBandwidthCtrlEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);
rtk_rate_igrBandwidthCtrlEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
rtk_rate_igrBandwidthCtrlRate_get(uint32 unit, rtk_port_t port, uint32 *pRate);
rtk_rate_igrBandwidthCtrlRate_set(uint32 unit, rtk_port_t port, uint32 rate);
rtk_rate_igrBandwidthFlowctrlEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);
rtk_rate_igrBandwidthFlowctrlEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
rtk_rate_igrBandwidthLowThresh_get(uint32 unit, uint32 *pLowThresh);
rtk_rate_igrBandwidthLowThresh_set(uint32 unit, uint32 lowThresh);
rtk_rate_portIgrBandwidthHighThresh_get(uint32 unit, rtk_port_t port, uint32 *pHighThresh);
rtk_rate_portIgrBandwidthHighThresh_set(uint32 unit, rtk_port_t port, uint32 highThresh);
rtk_rate_portIgrBandwidthCtrlExceed_get(uint32 unit,rtk_port_t port,uint32 *pIsExceed);
rtk_rate_portIgrBandwidthCtrlExceed_reset(uint32 unit, rtk_port_t port);
```

PROGRAMMING EXAMPLE

Ingress Bandwidth Control Setting and Enabled Ingress Bandwidth Flow Control Configurations

```
/* Enable bandwidth control and rate configured to 10Mbps */
port = 2;
rate = 625;    /* 625*16Kbps = 10Mbps */
rtk_rate_igrBandwidthCtrlEnable_set(unit, port, ENABLED);
rtk_rate_igrBandwidthCtrlRate_set(unit, port, rate);

/* Enable ingress bandwidth flow control function */
rtk_rate_igrBandwidthFlowctrlEnable_set(unit, port, ENABLED);
```

13.2 Traffic Admit

Ingress bandwidth function provides an option to admit particular traffic flow. It is a global option per traffic type. To admit a flow means the flow is not limited by ingress bandwidth module and it doesn't consume the configured-bandwidth.

The supported traffic types are listed below:

- ARPREQ_ADMIT for ARP request and Neighbor Solicitation for ICMPv6
- RMA_ADMIT for RMA packet
- BPDU_ADMIT for BPDU packet
- RTKPKT_ADMIT for the packet with ether type 0x8899 (RRCP、RLPP、RLDP…)
- IGMP_ADMIT for IGMP packet

Table 98: IGR_BWCTRL_CTRL Register

Field Name	Bits	Description
ARPREQ_ADMIT	1	Admit ARP Request and Neighbor Solicitation for ICMPv6 (Type:135) packet bypassing ingress bandwidth control. 1'b0: disable 1'b1: enable
RMA_ADMIT	1	Admit RMA (exclude BPDU, include PTP & LLDP) packet bypassing ingress bandwidth control. 1'b0: disable 1'b1: enable
BPDU_ADMIT	1	Admit BPDU packet bypassing ingress bandwidth control. 1'b0: disable 1'b1: enable
RTKPKT_ADMIT	1	Admit Realtek control packet (ethtype=0x8899) bypassing ingress bandwidth control. 1'b0: disable 1'b1: enable
IGMP_ADMIT	1	Admit IGMP packet bypassing ingress bandwidth control. Note: This includes IPv4 IGMP and IPv6 MLDv1/v2. 1'b0: disable 1'b1: enable

API REFERENCE

```
rtk_rate_stormControlBypass_get(uint32 unit, rtk_rate_storm_bypassType_t bypassType, rtk_enable_t
*pEnable);
rtk_rate_stormControlBypass_set(uint32 unit, rtk_rate_storm_bypassType_t bypassType, rtk_enable_t
enable);
```

PROGRAMMING EXAMPLE

Enabled IGMP Packet Bypass Ingress Bandwidth Control Configuration

```
/* Enable IGMP bypass feature */
bypassType = STORM_BYPASS_IGMP;
rtk_rate_stormControlBypass_set(unit, bypassType, ENABLED);
```

14 Storm Control

Storm control suppresses flooding traffic which prevents hosts connected to the switch from overwhelming. The device per ingress port supports storm control function to limit rate of unknown unicast traffic, unicast traffic (both known and unknown unicast), unknown multicast traffic, multicast (both known and unknown multicast), and broadcast traffic. The device also allows some specific protocol packet to pass through storm control filtering. Not only unknown storm control but also protocol storm control functions are supported to suppress BPDU, IGMP, and ARP request/Neighbor Solicitation traffic.

14.1 Unknown Storm Control

The device per ingress port supports storm control to suppress unicast, multicast, and broadcast traffic. The traffic that exceeds the limited rate is dropped. The limited rates are specified by UC_RATE (for unicast traffic), MC_RATE (for multicast traffic), and BC_RATE (for broadcast traffic) register fields. The storm control function is active whenever rate is not configured to 0xFFFFF.

PPS and BPS rate mode are both supported and is globally specified by register field MODE. If MODE is PPS, the unit of UC_RATE/MC_RATE/BC_RATE is packet per second; if MODE is BPS, the unit of UC_RATE/MC_RATE/BC_RATE is 16 Kbps. When rate mode is BPS, whether to count the IFG (Inter Frame Gap) is specified by a global register field INC_IFG.

The storm control utilizes leaky bucket mechanism, the configured rate could be considered as leaky rate. When ingress traffic reaches the bucket threshold, the packet is dropped. The thresholds are defined by per port register fields UC_BURST, MC_BURST, and BC_BURST. The unit of thresholds also depends on rate mode: if MODE is PPS, the unit is a packet; if MODE is BPS, the unit is a byte.

There are options for unicast and multicast traffic to decide whether to limit both known and unknown traffic or only limit unknown traffic:

- Unicast traffic
  - Limit unknown unicast traffic (UC_TYPE = 0)
  - Limit both known and unknown unicast traffic (UC_TYPE = 1)
- Multicast traffic (L2 and IP Multicast)
  - Limit unknown multicast traffic (MC_TYPE = 0)
  - Limit both known and unknown multicast traffic (MC_TYPE = 1)
- Broadcast traffic

For instance, to limit unknown multicast and broadcast traffic received from port 1 to 1000 packets per second and with 10 packets bucket threshold. It could be achieved by configuring MODE = 0 (packet based), MC_TYPE of port 1 = 0 (unknown multicast), MC_RATE = 1000, BC_RATE = 1000, MC_BURST = 10, and BC_BURST = 10.

Table 99: STORM_CTRL_CTRL Register

Field Name	Bits	Description
INC_IFG	1	Storm control rate include IFG(inter frame gap) and preamble. 1'b0: exclude 1'b1: include
MODE	1	Storm control is based on packet count or byte count. 1'b0: pps 1'b1: bps

Table 100: STORM_CTRL_PORT_UC/STORM_CTRL_PORT_MC/STORM_CTRL_PORT_BC Registers

Field Name	Bits	Description

UC_TYPE	1	Unicast stream type selection 1'b0: unknown unicast 1'b1: both known and unknown unicast
UC_RATE	20	Unknown unicast storm control rate for port n. In PPS mode, the unit is 1 PPS. For Giga and 10G port, the valid bit is [19:0]: 20'h0: blocking 20'hFFFFF: unlimited In BPS mode, the unit is 16Kbps. For Giga port, the valid bit is [15:0]: 20'h0: blocking 20'hFFFF: unlimited For 10G port, the valid bit is [19:0]: 20'h0: blocking 20'hFFFFF: unlimited
UC_BURST	16	Unknown unicast burst size Unit: packet in PPS or byte in BPS
MC_TYPE	1	Multicast Storm type selection. 1'b0: only unknown multicast 1'b1: both known and unknown multicast
MC_RATE	20	Multicast storm control rate for port n.
MC_BURST	16	Multicast burst size
BC_RATE	20	Broadcast storm control rate for port n.
BC_BURST	16	Broadcast burst size

API REFERENCE

```
rtk_rate_stormControlRateMode_get(uint32 unit,rtk_rate_storm_rateMode_t *pRate_mode);
rtk_rate_stormControlRateMode_set(uint32 unit,rtk_rate_storm_rateMode_t *rate_mode);
rtk_rate_stormControlRate_get(uint32 unit, rtk_port_t port, rtk_rate_storm_group_t storm_type, uint32
*pRate);
rtk_rate_stormControlRate_set(uint32 unit, rtk_port_t port, rtk_rate_storm_group_t storm_type, uint32
rate);
rtk_rate_stormControlTypeSel_get(uint32 unit, rtk_port_t port, rtk_rate_storm_group_t storm_type,
rtk_rate_storm_sel_t *pStorm_sel);
rtk_rate_stormControlTypeSel_set(uint32 unit, rtk_port_t port, rtk_rate_storm_group_t storm_type,
rtk_rate_storm_sel_t storm_sel);
rtk_rate_portStormControlBurstSize_get(uint32 unit, rtk_port_t port, rtk_rate_storm_group_t storm_type,
uint32 *pBurst_size);
rtk_rate_portStormControlBurstSize_set(uint32 unit, rtk_port_t port, rtk_rate_storm_group_t storm_type,
uint32 burst_size);
```

14.2 Protocol Packet Admission

Specific protocol packets should not be filtered by storm control or it may cause the protocol malfunction. Following registers are provided to admit specific protocol packet:

ARPREQ_ADMIT: applies to ARP Request and Neighbor Solicitation of ICMPv6 packets.
RMA_ADMIT: applies to packets with RMA DMAC (01-80-C2-00-00-XX), BPDU is exclued.
BPDU_ADMIT: applies to BPDU (01-80-C2-00-00-00).
RTKPKT_ADMIT: applies to Realtek control packets which EtherType is 0x8899.
IGMP_ADMIT: applies to IPv4 IGMP and IPv6 MLDv1/v2 packets.

Table 101: STORM_CTRL_CTRL Register

Field Name	Bits	Description
ARPREQ_ADMIT	1	Admit ARP Request and Neighbor Solicitation of ICMPv6 (Ethertype = 0x86DD, NextHeader = 0x3A, Type:135) packet bypassing storm control. 1'b0: disable 1'b1: enable
RMA_ADMIT	1	Admit RMA (exclude BPDU) packet bypassing storm control. 1'b0: disable 1'b1: enable
BPDU_ADMIT	1	Admit BPDU packet bypassing storm control. 1'b0: disable 1'b1: enable
RTKPKT_ADMIT	1	Admit Realtek control packet (ethtype=0x8899) bypassing storm control. 1'b0: disable 1'b1: enable
IGMP_ADMIT	1	Admit IPv4 IGMP and IPv6 MLDv1/v2packet bypassing storm control. 1'b0: disable 1'b1: enable

Note: Admitted packet would not consume the storm control bandwidth and never be filtered by storm control function.

API REFERENCE

*rtk_rate_stormControlBypass_get(uint32 unit, rtk_rate_storm_bypassType_t bypassType, rtk_enable_t *pEnable);*
*rtk_rate_stormControlBypass_get(uint32 unit, rtk_rate_storm_bypassType_t bypassType, rtk_enable_t *pEnable);*

14.3 Protocol Storm Control

In addition to unknown storm control, the device per ingress port also supports register fields BPDU_RATE, IGMP_RATE, and ARP_RATE to suppress BPDU, IGMP, and ARP Request/Neighbor Solicitation packets respectively. The protocol storm control function only supports PPS mode. If rate is configured as 0, all the specific control packets are dropped by the port. The function can be disabled by configuring rate as 0xFF.

Please be awared that protocol storm control and unknown storm control works concurrently and independently. That is, BPDU, IGMP, and ARP Request/Neighbor packet are examined by both protocol storm control module and unknown storm control module. The packet is dropped if one of the storm modules determines to drop it. Protocol packet admission described in previous section can be enabled to allow BPDU, IGMP, and ARP Request/Neighbor to pass through the unknown storm control module.

Table 102: STORM_CTRL_SPCL_PORT_RATE Register

Field Name	Bits	Description
BPDU_RATE	8	BPDU storm control rate for port n. Unit: PPS 8'h0: blocking 8'hFF: unlimited
IGMP_RATE	8	IGMP/MLD storm control rate for port n. Unit: PPS 8'h0: blocking

		8'hFF: unlimited
ARP_RATE	8	ARP REQUEST (IPv4) and Neighbor Solicitation (ICMPv6:Type 135) storm control rate for port n. Unit: PPS 8'h0: blocking 8'hFF: unlimited

API REFERENCE

*rtk_rate_stormControlProtoRate_get(uint32 unit, rtk_port_t port, rtk_rate_storm_proto_group_t storm_type, uint32*pRate);*
rtk_rate_stormControlProtoRate_set(uint32 unit, rtk_port_t port, rtk_rate_storm_proto_group_t storm_type, uint32 rate);

14.4 Overflow Indicator

A reaction, such as alarm may need to be taken for security reason once the strom has been detected. Thus, the deivce supports overflow indicators to record whether the storm ever occurs and the upper bound threshold was reached. The overflow indicator is per port per traffic type 1 bit register field and will not be cleared unless write.

Table 103: Storm Control Overflow Registers

Register Name	Field Name	Bits	Description
STORM_CTRL_PORT_BC_EXCEED_FLG	BC_EXCEED_FLG	1	Broadcast storm control overflow indicator. Write 1 to clear. 1'b0: not overflow 1'b1: overflow
STORM_CTRL_PORT_MC_EXCEED_FLG	MC_EXCEED_FLG	1	Multicast storm control overflow indicator. Write 1 to clear. 1'b0: not overflow 1'b1: overflow
STORM_CTRL_PORT_UC_EXCEED_FLG	UC_EXCEED_FLG	1	Unicast storm control overflow indicator. Write 1 to clear. 1'b0: not overflow 1'b1: overflow
STORM_CTRL_SPCL_PORT_BPDU_EXCEED_FLG	BPDU_EXCEED_FLG	1	BPDU storm control overflow indicator. Write 1 to clear. 1'b0: not overflow 1'b1: overflow
STORM_CTRL_SPCL_PORT_IGMP_EXCEED_FLG	IGMP_EXCEED_FLG	1	IGMP storm control overflow indicator. Write 1 to clear. 1'b0: not overflow 1'b1: overflow
STORM_CTRL_SPCL_PORT_ARP_EXCEED_FLG	ARP_EXCEED_FLG	1	ARP storm control overflow indicator. Write 1 to clear. 1'b0: not overflow 1'b1: overflow

API REFERENCE

*rtk_rate_stormControlExceed_get(uint32 unit, rtk_port_t port, rtk_rate_storm_group_t storm_type, uint32 *pIsExceed);*
rtk_rate_stormControlExceed_reset(uint32 unit, rtk_port_t port, rtk_rate_storm_group_t storm_type);
*rtk_rate_stormControlProtoExceed_get(uint32 unit, rtk_port_t port, rtk_rate_storm_proto_group_t storm_type, uint32 *pIsExceed);*
rtk_rate_stormControlProtoExceed_reset(uint32 unit, rtk_port_t port, rtk_rate_storm_proto_group_t storm_type);

15 802.1X

In many environments, the access to the service offered by LAN is permitted only after authorized for the security consideration. 802.1X is a port-based network access control protocol that provides a mean of authenticating and authorizing devices attached to the LAN port, and of preventing access to that port if the authentication process fails. The following figure illustrates the infrastructure of 802.1X:

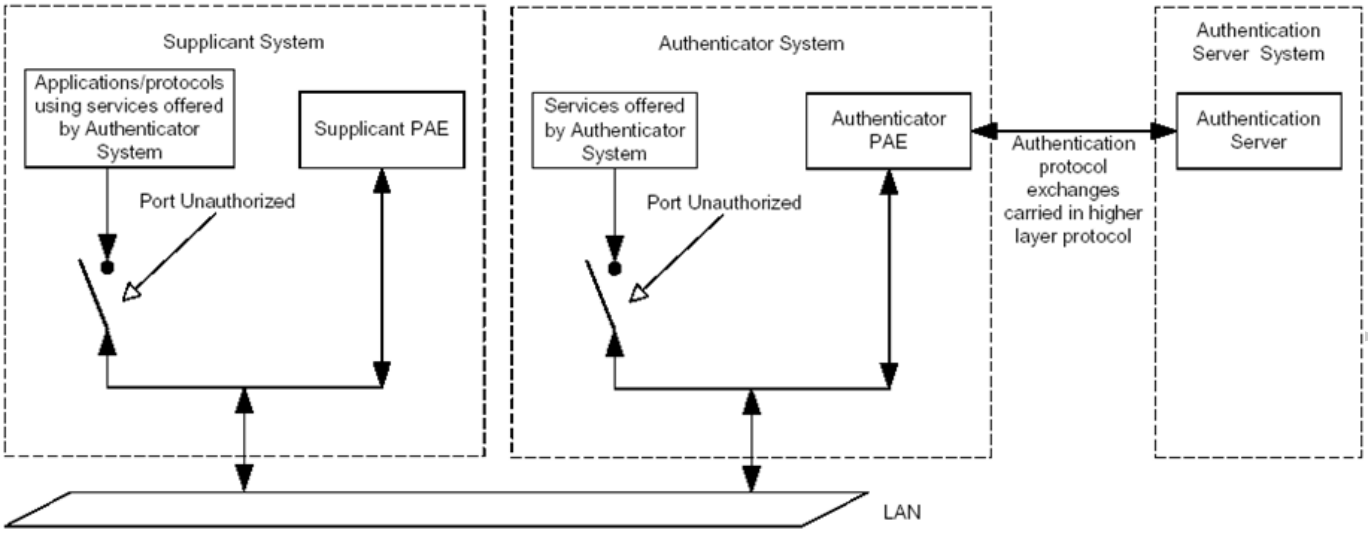


Figure 30: Authenticator, Supplicant, and Authentication Server Roles

In the infrastructure, there are three roles – authenticator, supplicant, and authentication server. In this document, the authenticator is usually the switch which works as a communication agent between supplicant (such as PC) and authentication server (may be a security database). The supplicant provides credentials to the authentication server through authenticator. If the authentication server agrees this supplicant, it will send “ACCEPT” message to the switch, otherwise it will send “REJECT”. On receiving the above decision from authentication server, the switch will open/close its LAN service to the supplicant.

For controlling the network access, there is a premise that an unauthorized host must not “register” its source MAC in switch’s L2 table. Therefore, by controlling the L2 learning and forwarding behavior, a host may be allowed or denied to access the LAN service.

This chapter suggests ways for trapping above communication packets and how to provide/close LAN network service making use of ASIC’s facilities. The state machine transformation and encapsulation/de-capsulation of packet is defined in 802.1X protocol and out of the scope of this document. For detail behavior of this protocol, please refer to IEEE 802.1X standard.

15.1 Port-based Authentication

In port-based authentication, a port on the switch is permitted to use network service only after successful authentication. Due to its port-based essence, all hosts connect to this port are view as authorized.

15.1.1 Receiving Control Packet

In port-based authentication, the communication packets between supplicant and switch are known as EAPOL and those between switch and authentication server are known as EAP on RADIUS. If 802.1X is enabled on a port, both

EAPOL and RADIUS packets should be trapped to CPU.
To trap EAPOL, configuration register SPCL_TRAP_EAPOL_CTRL can be used.

Table 104: SPCL_TRAP_EAPOL_CTRL Register

Field Name	Bits	Description
ACT	1	Trap EAPOL (EtherType=0x888E) packet to CPU. 1'b0: forward 1'b1: trap

Note that whenever this bit is set to enabled, the EAPOL packets will be trapped to CPU even they are received on an 802.1X disabled port. If trap is not desirous on disabled port, please use ACL to trap EAPOL packet instead (ACL can specify the ports to apply).
EAP on RADIUS packet is a typical UDP packet, therefore it can be trapped by ACL which qualifies the UDP port.

API REFERENCE

```
rtk_trap_mgmtFrameAction_get (uint32 unit, rtk_trap_mgmtType_t frameType, rtk_action_t *pAction);
rtk_trap_mgmtFrameAction_set(uint32 unit, rtk_trap_mgmtType_t frameType, rtk_action_t action);
```

15.1.2 Control on Network Access

To prevent supplicant from access the network service illegally, configuration register L2_PORT_SALRN and L2_PORT_NEW_SA_FWD can be used. When a port is 802.1X enabled, takes the following steps to control the network access on this port:

1. Flush all unicast MACs learnt on this port
2. Configure L2 table L2_PORT_SALRN to "Not Learn" for a packet whose SA is new to device
3. Configure L2_PORT_NEW_SA_FWD to "Drop" for a packet whose SA is new to device

By flushing all MACs learnt on this port, all incoming packets are treated as new SA and thus be dropped before this port is authorized. After successful authentication, we change the learning mode of this port to "ASIC Auto Learn" by L2_PORT_SALRN and forwarding mode to "Forward" by L2_PORT_NEW_SA_FWD. Afterward, all ingress traffic is forwarded normally.

Table 105: L2_PORT_NEW_SALRN Register

Field Name	Bits	Description
L2_PORT_NEW_SALRN	2	SA learning behavior 00b: ASIC Auto Learn 01b: learn as suspend 10b: not Learn 11b: reserved
L2_PORT_NEW_SA_FWD	2	Packet forwarding behavior when received packet with new SMAC 0b00: forward 0b01: drop 0b10: trap 0b11: copy to CPU

API REFERENCE

```
rtk_l2_newMacOp_get(
    uint32                  unit,
    rtk_port_t              port,
    rtk_l2_newMacLrnMode_t  *pLrnMode,
    rtk_action_t            *pFwdAction);
```

```
rtk_l2_newMacOp_set(
    uint32                      unit,
    rtk_port_t                  port,
    rtk_l2_newMacLrnMode_t   lrnMode,
    rtk_action_t                fwdAction);
```

15.1.3 OperControlledDirections

The OperControlledDirections is a parameter defined in the IEEE 802.1X standard which defines whether an authorized port can talk to an unauthorized port:

1. **OperControlledDirections = Both**. An authorized port can NOT talk to an unauthorized port.
2. **OperControlledDirections = In**. An authorized port can talk to an unauthorized port.

In normal switch forwarding process, the destination of a packet may exist or not exist in L2 table. If the destination (DMAC) is already in L2 table, it implies the destination host has been authorized (or the egress port is 802.1X disabled). In another situation that DMAC is unknown, this packet will be flooded to all port within VLAN. Due to the principle that a host attached to an unauthorized port must not register its source MAC address in L2 table, we can accomplish this functionality by properly setting the L2 lookup miss flooding portmask (includes L2 unicast, L2 multicast, IPv4/IPv6 multicast and broadcast flooding portmask):

$FLD\_PMSK = \{disable, port\text{-}auth, \{port\text{-}unAuth \cap opdir[port] = IN\}\}$

When a port is 802.1X enabled and OperControlledDirections set to “Both”, it should be removed from the *FLD_PMSK* before authorized. Otherwise, this port should be put to *FLD_PMSK* again.

API REFERENCE

```
rtk_l2_portLookupMissAction_get(uint32 unit, rtk_port_t port, rtk_l2_lookupMissType_t type, rtk_action_t
*pAction);
rtk_l2_portLookupMissAction_set(uint32 unit, rtk_port_t port, rtk_l2_lookupMissType_t type, rtk_action_t
action);
rtk_l2_lookupMissFloodPortMask_get(uint32 unit, rtk_l2_lookupMissType_t type, rtk_portmask_t
*pFlood_portmask);
rtk_l2_lookupMissFloodPortMask_add(uint32 unit, rtk_l2_lookupMissType_t type, rtk_port_t flood_port);
rtk_l2_lookupMissFloodPortMask_del(uint32 unit, rtk_l2_lookupMissType_t type, rtk_port_t flood_port);
```

15.2 MAC-based authentication

Besides port-based authentication, the MAC-based authentication on a per-port configured basis can also be supported. In port-based authentication, if a port got authorized, all hosts connected to which are permitted to access network. But in MAC-based authentication, only the host with authorized source MAC could access the LAN service.

15.2.1 Receiving Control Packet

When a port is configured to MAC-based authentication enabled, depending on the application, the MAC-based authentication process may have or may not have EAPOL control packet. If EAPOL is used as the control packet, the trapped configuration is same as previous description. If the application uses the data packets credentials, L2_PORT_NEW_SALRN can be used to trap the packets.

If whole authentication process only needs the first packet, L2_PORT_NEW_SALRN should be set to “learn as suspend” for just trapping the first packet of the same SMAC to CPU and drop others in a limited time period

(depends on the L2 aging-time configuration). If the authentication process needs many packets, L2_PORT_NEW_SALRN should be set to “Not Learn”. For both above scenarios, L2_PORT_NEW_SA_FWD should be set to “Trap”.

If administrator decides to disable MAC-based authentication on this port, just set the L2_PORT_NEW_SALRN and L2_PORT_NEW_SA_FWD to “ASIC Auto Learn” and “Forward” respectively.

API REFERENCE

```
rtk_l2_newMacOp_get(
    uint32                      unit,
    rtk_port_t                  port,
    rtk_l2_newMacLrnMode_t   *pLrnMode,
    rtk_action_t                *pFwdAction);
rtk_l2_newMacOp_set(
    uint32                      unit,
    rtk_port_t                  port,
    rtk_l2_newMacLrnMode_t   lrnMode,
    rtk_action_t                fwdAction);
```

15.2.2 Control on Network Access

Once the L2_PORT_NEW_SALRN is set to “learn as suspend”, ASIC will drop all following packets except the first one. Thus prevent from unauthorized access. If L2_PORT_NEW_SALRN is set to “Trap”, all unauthorized incoming packets can be dropped by CPU.

If the host passes the authentication criteria, a corresponding L2 entry should be inserted to the device. Then, all traffic from this host is granted to the system and LAN services are still unavailable to other hosts.

API REFERENCE

```
rtk_l2_addr_add(uint32 unit, rtk_l2_ucastAddr_t *pL2_addr);
rtk_l2_addr_get(uint32 unit, rtk_l2_ucastAddr_t *pL2_addr);
rtk_l2_addr_set(uint32 unit, rtk_l2_ucastAddr_t *pL2_addr);
rtk_l2_addr_del(uint32 unit, rtk_vlan_t vid, rtk_mac_t *pMac);
```

15.2.3 OperControlledDirections

Unlike port-based authentication, the OperControlledDirections in MAC-based authentication is global setting. And the L2 lookup miss flooding portmask can be utilized again.

- OperControlledDirections = Both, `FLD_PMSK = { ALL_port }`
- OperControlledDirections = In, `FLD_PMSK = { ALL_port - macBasedEnable[port]}`

An authorized host must have registered its MAC address in L2 table, so just to remove the ports that enable MAC-based authentication when OperControlledDirections is configured as “IN” mode.

15.3 Port-based Guest VLAN

Guest VLAN is an environment that provides unauthorized host limited network service. The device can easily support the port-based Guest VLAN.

When a port failed in authentication, we could put this port to Guest VLAN by set its port-based VLAN ID and move the port out from its original VLAN to Guest VLAN. By this way, the guest traffic is separated from normal traffic.

As soon as the port is authenticated, the VLAN member configuration should be restored back to original.

PROGRAMMING EXAMPLE

```
/* Set port-based unauthorized packet action */

rtk_l2_newMacLrnMode_t oldsa_mode;
rtk_action_t old_action;

tk_l2_newMacOp_get(physicPort.dev, physicPort.port, &oldsa_mode, &old_action);
rtk_l2_newMacOp_set(physicPort.dev, physicPort.port, oldsa_mode, action);

/* store the configuration for conveniently */
portUnauthAct[port] = action;

/* Set rate limit for trap2cpu action */
```

```
/* Set port-based authentication status */

switch (status)
{
    case SYS_AUTH_DISABLE:
    case SYS_AUTH:
        rtk_l2_newMacOp_set(physicPort.dev, physicPort.port,
                                                HARDWARE_LEARNING,
                                                ACTION_FORWARD);
        break;

    case SYS_UNAUTH:
    default:
    {
        rtk_l2_flushCfg_t config;

        osal_memset(&config, 0, sizeof(rtk_l2_flushCfg_t));
        config.flushByPort = TRUE;
        config.portOrTrunk = TRUE; /* port-based */
        config.port = physicPort.port;

        rtk_l2_newMacOp_get(port, &action));

        rtk_l2_newMacOp_set(physicPort.dev, physicPort.port,
                                                NOT_LEARNING,
                                                action);
        rtk_l2_ucastAddr_flush(physicPort.dev, &config);
        break;
    }
}

/* store the configuration for conveniently */
portAuthEbl[port] = status;
```

16 Attack Prevention

The system support hardware Attack Checker to filter the malicious packets. Attacker attempts to make a machine or network resource unavailable to its intended users. The checker can detect these packets and filter them to protect the receiver. Or trap these attack packets to CPU for analysis.

16.1 Attack Detection

The device supports the following types of attack prevention.

Table 106: Attack Types

Type	Description	Action
DA_EQUAL_SA	Source MAC address is equal to Destination MAC Address	Drop/Trap
LAND	IPv4/IPv6 source address is equal to destination address	Drop/Trap
UDP_BLAT	UDP packet with source port = destination port	Drop/Trap
TCP_BLAT	TCP packet with source port = destination port	Drop/Trap
POD	Ping-of-Death, (IP payload length + fragment offset) > 65535.	Drop/Trap
IPV6_FRAG_LEN_MIN	Fragment size of IPv6 datagram < IPV6.FRAG_LEN_MIN (Range: 0~65535)	Drop/Trap
ICMP_FRAG_PKT	ICMP(v4/v6) protocol data are carried in a fragmented IP(v4/v6) datagram.	Drop/Trap
ICMPV4_PING_MAX	(Total length field – header length) > ICMP.PKT_LEN_MAX (Range: 0~65535)	Drop/Trap
ICMPV6_PING_MAX	Payload length > ICMP.PKT_LEN_MAX (Range: 0~65535)	Drop/Trap
SMURF	ICMP(v4/v6) echo request packet with broadcast address	Drop/Trap
TCP_HDR_LEN_MIN	TCP header length < TCP.HDR_LEN_MIN (Range: 0~31)	Drop/Trap
SYN_SPORT_LESS_1024	TCP Control flag SYN=1 & ACK=0 & TCP SPORT<1024	Drop/Trap
NULL_SCAN	TCP Seq_Num = 0, All control flag are zeroes.	Drop/Trap
XMAS	TCP Seq_Num = 0, Control flag (FIN, URG, PSH are set)	Drop/Trap
SYN_FIN	TCP Control flag (SYN, FIN are set)	Drop/Trap
SYN_RST	TCP Control flag (SYN, RST are set)	Drop/Trap
TCP_FRAG_OFF_MIN	TCP packet with IP fragment offset = 1 (means 8Bytes)	Drop/Trap

Packet goes over the functional blocks in below sequence.

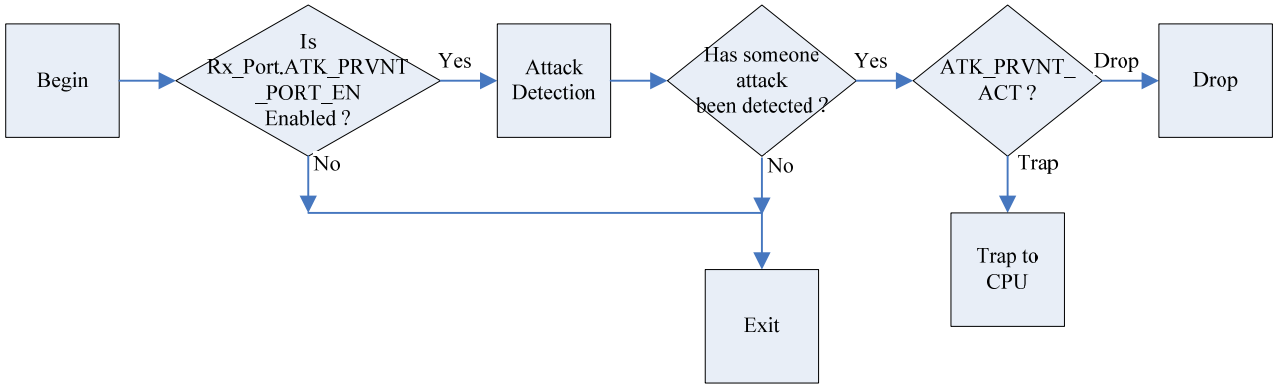


Figure 31: Attack Prevention Packet Flow

Table 107: ATK_PRVNT_PORT_EN Register

Field Name	Bits	Description
EN	1	Enable per-port attack prevention.

Table 108: ATK_PRVNT_CTRL Register

Field Name	Bits	Description
DA_EQUAL_SA	1	Filter packets with MACDA = MACSA.
LAND	1	Filter IPv4/IPv6 packets whose SIP = DIP.
UDP_BLAT	1	Filter IPv4/IPv6 UDP packets whose SPORT = DPORT.
TCP_BLAT	1	Filter IPv4/IPv6 TCP packets whose SPORT = DPORT.
POD	1	Filter IPv4/IPv6 packets that length is larger than 64K bytes through fragments. (Ping of Death)
IPV6_FRAG_LEN_MIN	1	Check for minimum size of IPv6 fragments. (If packet length is larger than ATK_PRVNT_IPV6_CTRL.FRAG_LEN_MIN)
ICMP_FRAG_PKT	1	Filter fragmented ICMP packets (IPv4 & IPv6).
ICMPV4_PING_MAX	1	Filter ICMPv4 ping packets with payload size greater than the programmable value in ATK_PRVNT_ICMP_CTRL.PKT_LEN_MAX.
ICMPV6_PING_MAX	1	Filter ICMPv6 ping packets with payload size greater than the programmable value in ATK_PRVNT_ICMP_CTRL.PKT_LEN_MAX.
SMURF	1	Filter ICMP packets of ping request to a broadcast DIP (x.x.x.255) and the SIP is spoofed (victim).
TCP_HDR_LEN_MIN	1	Check first TCP fragmented packet that don't have the full TCP header. (i.e., data offset field in TCP header should have the value larger than or equal to the configured value in ATK_PRVNT_TCP_CTRL.HDR_LEN_MIN.
SYN_SPORT_LESS_1024	1	Filter TCP packets with SYN bit set and ACK bit not set and sport less than 1024.
NULL_SCAN	1	Filter TCP packets with sequence number is zero and the control-bits are zeroes.
XMAS	1	Filter TCP packets with sequence number is zero and the FIN,URG, and PSH bits are set (XMAS Scan attack).
SYN_FIN	1	Filter TCP packets with SYN and FIN set (SYN-FIN Scan attack).
SYN_RST	1	Filter TCP packets with SYN and RST set (SYN-RST Scan attack).
TCP_FRAG_OFF_MIN	1	Filter TCP non-initial fragments carrying TCP header.

Table 109: ATK_PRVNT_ACT Register

Field Name	Bits	Description
DA_EQUAL_SA	1	Determine the action for the DA=SA attack packets. 0b0: drop 0b1: trap
LAND	1	Determine the action for the LAND attack packets. 0b0: drop 0b1: trap
UDP_BLAT	1	Determine the action for the UDP blat attack packets. 0b0: drop 0b1: trap
TCP_BLAT	1	Determine the action for the TCP blat attack packets. 0b0: drop 0b1: trap
POD	1	Determine the action for the POD attack packets.

		0b0: drop 0b1: trap
IPV6_FRAG_LEN_MIN	1	Determine the action for the IPv6 minimum fragmented packets. 0b0: drop 0b1: trap
ICMP_FRAG_PKT	1	Determine the action for the ICMP fragmented packets. 0b0: drop 0b1: trap
ICMPV4_PING_MAX	1	Determine the action for the ICMPv4 ping attack packets. 0b0: drop 0b1: trap
ICMPV6_PING_MAX	1	Determine the action for the ICMPv6 ping attack packets. 0b0: drop 0b1: trap
SMURF	1	Determine the action for the SMURF attack packets. 0b0: drop 0b1: trap
TCP_HDR_LEN_MIN	1	Determine the action for the TCP header too small packets. 0b0: drop 0b1: trap
SYN_SPORT_LESS_1024	1	Determine the action for the SYN_SPORT_LESS_1024 packets. 0b0: drop 0b1: trap
NULL_SCAN	1	Determine the action for the NULLSCAN packets. 0b0: drop 0b1: trap
XMAS	1	Determine the action for the XMAS packets. 0b0: drop 0b1: trap
SYN_FIN	1	Determine the action for the SYNFIN packets. 0b0: drop 0b1: trap
SYN_RST	1	Determine the action for the SYNRST packets. 0b0: drop 0b1: trap
TCP_FRAG_OFF_MIN	1	Determine the action for the TCP_FRAG_OFF_MIN packets. 0b0: drop 0b1: trap

Table 110: ATK_PRVNT_IPV6_CTRL Register

Field Name	Bits	Description
FRAG_LEN_MIN	16	Value to use when checking for minimum size of IPV6 fragments. (Checking is enabled by setting APCR.IPV6_MIN_FRAG_LEN).

Table 111: ATK_PRVNT_ICMP_CTRL Register

Field Name	Bits	Description
PKT_LEN_MAX	16	Maximum length ICMP packet allowed before dropping.

Table 112: ATK_PRVNT_TCP_CTRL Register

Field Name	Bits	Description
HDR_LEN_MIN	5	Minimum TCP header length allowed. (minimum 0 bytes, maximum 31 bytes)

Table 113: ATK_PRVNT_SMURF_CTRL Register

Field Name	Bits	Description
NETMASK	32	Bitmask of Network Mask to check the Smurf.

API REFERENCE

```
rtk_sec_portAttackPreventEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);
rtk_sec_portAttackPreventEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
rtk_sec_attackPreventAction_get(uint32 unit, rtk_sec_attackType_t attack_type, rtk_action_t *pAction);
rtk_sec_attackPreventAction_set(uint32 unit, rtk_sec_attackType_t attack_type, rtk_action_t action);

rtk_sec_minIPv6FragLen_get(uint32 unit, uint32 *pLength);
rtk_sec_minIPv6FragLen_set(uint32 unit, uint32 length);
rtk_sec_maxPingLen_get(uint32 unit, uint32 *pLength);
rtk_sec_maxPingLen_set(uint32 unit, uint32 length);
rtk_sec_minTCPHdrLen_get(uint32 unit, uint32 *pLength);
rtk_sec_minTCPHdrLen_set(uint32 unit, uint32 length);
rtk_sec_smurfNetmaskLen_get(uint32 unit, uint32 *pLength);
rtk_sec_smurfNetmaskLen_set(uint32 unit, uint32 length);
```

16.2 Validation Check

The device supports to check the validation of ARP packet.

16.2.1 Invalid ARP Packet

Below table describes the criteria for determining an invalid ARP packet.

Table 114: Invalid ARP Packet

Type	Description	Action
ARP_INVLD	Invalid IPv4 ARP packet, that is matched the one of the following conditions: 1. For request packet: (1) MAC SA != ARP Sender MAC Address (2) ARP Sender IP = all-zero or multicast or broadcast IP address. 2. For Reply Packet: (1) MAC SA != ARP Sender MAC Address. (2) ARP Target MAC Address = all-zero or multicast address. (3) ARP Target MAC Address != MAC DA. (4) ARP Sender IP = all-zero or multicast or broadcast IP address. (5) ARP Target IP = all-zero or multicast or broadcast IP address.	Forward/Drop/Trap

Table 115: ATK_PRVNT_ARP_INVLD_PORT_ACT Register

Field Name	Bits	Description
ACT	1	Action for ARP invalid packets.

	0b00: Forward 0b01: Drop 0b10: Trap 0b11: Reserved

API REFERENCE

```
rtk_sec_portAttackPrevent_get(
    uint32                  unit,
    rtk_port_t              port,
    rtk_sec_attackType_t    attack_type,
    rtk_action_t            *pAction);
rtk_sec_portAttackPrevent_set(
    uint32                  unit,
    rtk_port_t              port,
    rtk_sec_attackType_t    attack_type,
    rtk_action_t            action);
```

16.2.2 Gratuitous ARP Packet

Below table describes the criteria for determining a gratuitous ARP packet.

Table 116: Gratuitous ARP Packet

Type	Description	Action
GRATUITOUS_ARP	Gratuitous ARP packet: For ARP request packet: (Target_HW_ADDR = all-zeroes or broadcast) AND (Sender_PROTO_ADDR = Target_PROTO_ADDR). For ARP reply packet: (Sender_HW_ADDR = Target_HW_ADDR) AND (Sender_PROTO_ADDR = Target_PROTO_ADDR)	Forward/Drop/Trap

Table 117: ATK_PRVNT_PORT_GARP_ACT Register

Field Name	Bits	Description
GARP_ACT	1	Action for Gratuitous ARP packets. 0b00: Forward 0b01: Drop 0b10: Trap 0b11: Reserved

API REFERENCE

```
rtk_sec_portAttackPrevent_get(
    uint32                  unit,
    rtk_port_t              port,
    rtk_sec_attackType_t    attack_type,
    rtk_action_t            *pAction);
rtk_sec_portAttackPrevent_set(
    uint32                  unit,
    rtk_port_t              port,
```

```
rtk_sec_attackType_t      attack_type,
rtk_action_t              action);
```

16.3 Notes

1. The default configuration is not to filter any types of attack packets.
2. The MAC address of the packets dropped by Attack Prevention function will not be learnt.
3. The drop priority of Attack Prevention is higher than the trap priority of Ingress ACL.
4. The trap priority of Attack Prevention is lower than the drop priority of Ingress ACL.

17 Priority Decision

Quality of Service (QoS) is a technology for managing network traffic, and provides guarantee bandwidth on ability traffic to network applications or protocols. In IEEE 802.1Q Standard, eight priorities (0~7) are defined and are used to map traffic classes. The device provides eight traffic classes, and eight priorities which can be one-to-one mapping to traffic classes.

17.1 Priority Assignment

Traffic is classified into different traffic classes according to its properties. Seven priority sources are supported by the device and each priority source represents a particular property. The priority sources are listed below:

- Port
- Ingress ACL
- DSCP
- Inner-Tag
- Outer-Tag
- Mac-based/IP-subnet-based VLAN
- Protocol-and-port-based VLAN

The seven priority sources decide a unique 3-bit Internal Priority through Priority Decision Table for each receiving packet. The Internal Priority is then mapped to a traffic class using a global Internal Priority to QID Mapping Table. The flow how a packet is classified into traffic class is shown in below figure.

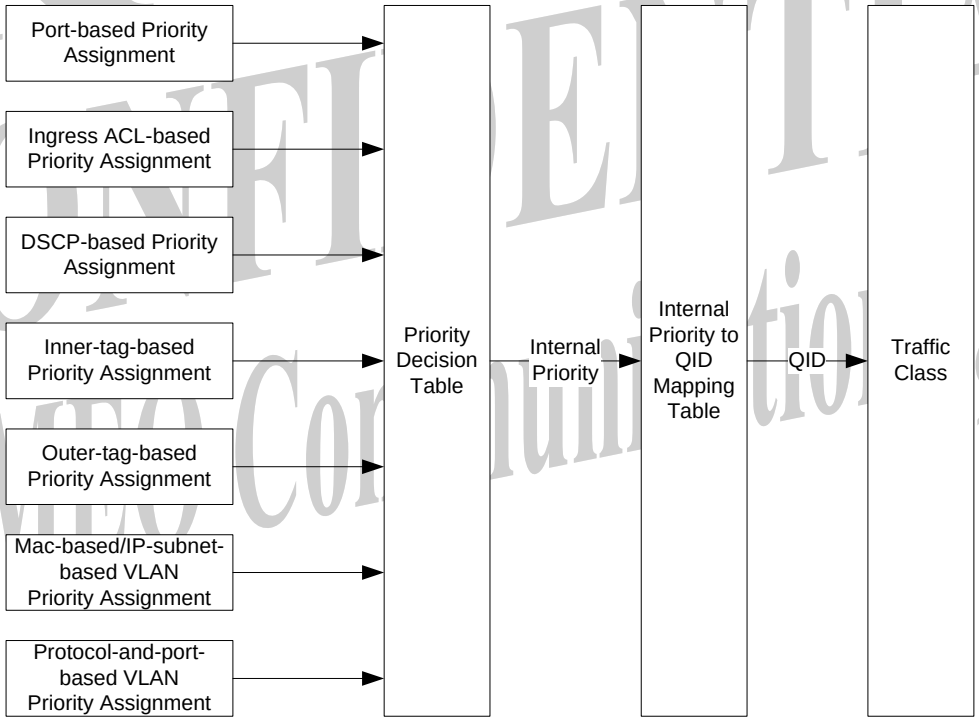


Figure 32: Priority Assignment Flow Chart

Note: The Internal Priority is only used to map traffic class but not encapsulated to the packet before transmitting. For the priority of a transmitting packet, please refer to Egress Remarking chapter.

17.1.1 Port Priority Assignment

The device supports 3-bit internal priority configuration register PRI_SEL_PORT_PRI for each ingress port.

Table 118: PRI_SEL_PORT_PRI Register

Field Name	Bits	Description
INTPRI_PORT	3	Port-based internal priority.

API REFERENCE

```
rtk_qos_portPri_get(uint32 unit, rtk_port_t port, rtk_pri_t *pInt_pri);
rtk_qos_portPri_set(uint32 unit, rtk_port_t port, rtk_pri_t int_pri);
```

17.1.2 Ingress ACL Priority Assignment

Incoming packets are examined by ingress ACL before going to priority decision module. Ingress ACL priority source is NULL if the packet is not assigned an internal priority by Ingress ACL module.

17.1.3 DSCP Priority Assignment

DSCP priority source is NULL if packet is neither IPv4 nor IPv6 packet. 8-bit Traffic Class filed in IPv6 header is equal to the Type Of Services (TOS) byte in IPv4 header. DSCP is in MSB 6-bit of TOS field, and it is remapped to internal priority by configuration register PRI_SEL_DSCP_REMAP.

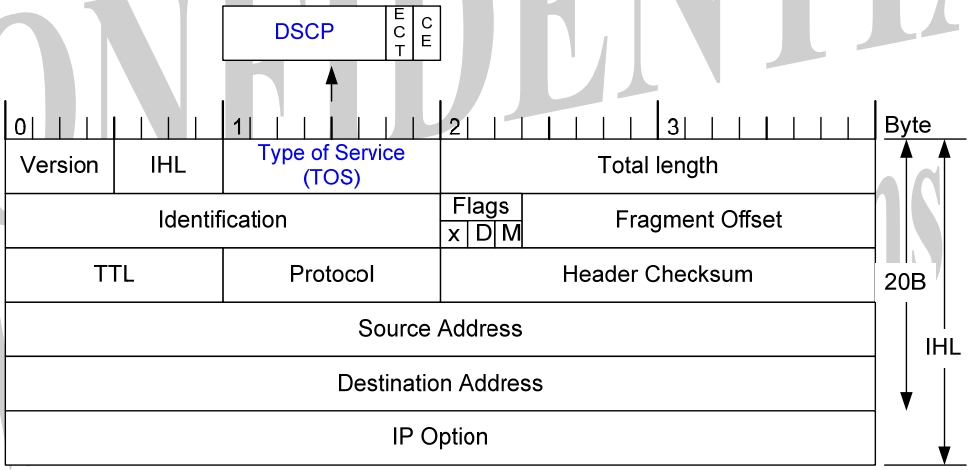


Figure 33: IPv4 Header Format

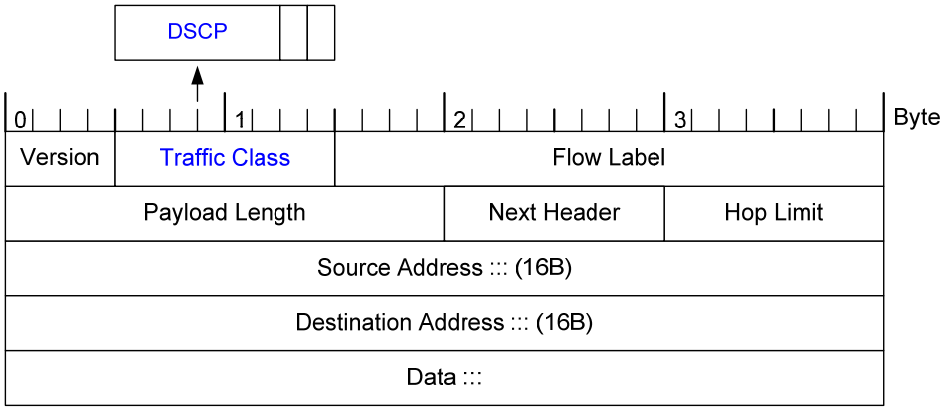


Figure 34: IPv6 Header Format

In addition to DSCP remapping, the device supports an invalid DSCP configuration register field PRI_SEL_CTRL.DSCP_INVLD_VAL. If DSCP hits the configuration, DSCP priority source is treated as NULL.

Table 119: PRI_SEL_DSCP_REMAP Register

Field Name	Bits	Description
INTPRI_DSCP	3	Remap DSCP to internal priority.

Table 120: PRI_SEL_CTRL Register

Field Name	Bits	Description
DSCP_INVLD_EN	3	Invalidate specific DSCP value. 1'b0: disabled 1'b1: enabled
DSCP_INVLD_VAL	6	Invalidate specific DSCP value. If DSCP value matches the configuration, system treats DSCP priority source as NULL.

API REFERENCE

```
rtk_qos_dscpPriRemap_get(uint32 unit, uint32 dscp, rtk_pri_t *pInt_pri);
rtk_qos_dscpPriRemap_set(uint32 unit, uint32 dscp, rtk_pri_t int_pri);
rtk_qos_invldDscpEnable_get(uint32 unit, rtk_enable_t *pEnable);
rtk_qos_invldDscpEnable_set(uint32 unit, rtk_enable_t enable);
rtk_qos_invldDscpVal_get(uint32 unit, uint32 *pDscp);
rtk_qos_invldDscpVal_set(uint32 unit, uint32 dscp);
```

17.1.4 Inner-Tag Priority Assignment

If packet hits the ingress VLAN conversion table for C → C' conversion, the priority C2SC_NEWPRI is used for mapping internal priority. Otherwise, the priority of inner-tag is used. The Inner-Tag priority source is NULL if packet is not an inner tag packet.

API REFERENCE

```
rtk_qos_1pPriRemap_get(uint32 unit, rtk_pri_t dot1p_pri, rtk_pri_t *pInt_pri);
rtk_qos_1pPriRemap_set(uint32 unit, rtk_pri_t dot1p_pri, rtk_pri_t int_pri);
```

17.1.5 Outer-Tag Priority Assignment

If packet is an outer tag packet, outer tag priority is remapped to internal priority by configuration register PRI_SEL_OPRI_DEI0_REMAP/PRI_SEL_OPRI_DEI1_REMAP. PRI_SEL_OPRI_DEI0_REMAP is used for the packet with DEI 0 while PRI_SEL_OPRI_DEI1_REMAP is used for the packet with DEI 1. The Outer-Tag priority source is NULL if packet is not an outer tag packet.

Table 121: PRI_SEL_OPRI_DEI0_REMAP Register

Field Name	Bits	Description
INTPRI_OPRI	3	Remap outer-tag priority with DEI 0 to internal priority.

Table 122: PRI_SEL_OPRI_DEI1_REMAP Register

Field Name	Bits	Description
INTPRI_OPRI	3	Remap outer-tag priority with DEI 1 to internal priority.

API REFERENCE

```
rtk_qos_outer1pPriRemap_get(uint32 unit, rtk_pri_t dot1p_pri, uint32 dei, rtk_pri_t *pInt_pri);
rtk_qos_outer1pPriRemap_set(uint32 unit, rtk_pri_t dot1p_pri, uint32 dei, rtk_pri_t int_pri);
```

17.1.6 MAC-based/IP-subnet-based VLAN Priority Assignment

MAC-based VLAN/IP Subnet-based VLAN priority is only applicable for untag packet. MAC-based/IP-subnet-based VLAN priority source is NULL if the packet doesn't be assigned an internal priority by MAC-based/IP-subnet-based VLAN table.

17.1.7 Protocol-and-port-based VLAN Priority Assignment

Protocol-and-Port-based VLAN priority is only applicable for untag packet. Protocol-and-port-based VLAN priority source is NULL if the packet doesn't be assigned an internal priority by Protocol-and-port-based VLAN.

17.2 Priority Selection Table

A packet could have at most seven different priorities from Ingress Port, Ingress ACL, DSCP, Inner-Tag, Outer-Tag, MAC-based/IP-subnet-based VLAN and Protocol-and-port-based VLAN priority assignments. One of them is selected to be unique internal priority for incoming packet according to the Priority Selection Table. The device supports four Priority Selection Table configurations (PRI_SEL_TBL_CTRL register) and per-ingress-port can bind to one of these four Priority Selection Tables through PRI_SEL_PORT_TBL_IDX_CTRL.

Weight value 0~7 can be configured to each priority source in Priority Selection Table. Weight 0 stands for ignoring the priority source. If all priority source weights are configured to 0, the internal priority produced by Priority Selection Table would be always 0. Valid weight values should be configured to 1~7, while 7 is the highest weight. Highest priority is taken to be the internal priority while more than one priority sources having the same weight configurations.

Table 123: PRI_SEL_TBL_CTRL Register

Field Name	Bits	Description
PORT_WT	3	Priority selection weight of Port priority.

		3'b000: ignore the priority source 3'b001 ~ 3'b111: weight value 1 ~ 7. 7 is the highest weight.
ING_ACL_WT	3	Priority selection weight of Ingress ACL priority.
DSCP_WT	3	Priority selection weight of DSCP priority.
ITAG_WT	3	Priority selection weight of Inner-Tag priority.
OTAG_WT	3	Priority selection weight of Outer-Tag priority.
MAC_VLAN_WT	3	Priority selection weight of MAC-based/IP-subnet-based VLAN priority.
PROTO_VLAN_WT	3	Priority selection weight of Protocol-and-port-based VLAN priority.

Table 124: PRI_SEL_PORT_TBL_IDX_CTRL Register

Field Name	Bits	Description
TBL_IDX	2	Specify the priority selection table to bind for the ingress port.

Take below figure for example. Weights of Port and Ingress ACL are 0, therefore the priority determined by Port and Ingress ACL are ignored. Protocol-and-port-based VLAN and Outer-Tag have the highest weight value 7, but both their priority results are NULL. Continue to check reset priority sources, DSCP, Inner-Tag and Mac-based/IP-subnet-based VLAN have same weight 6. Thus, the highest priority among them is 3 and is taken as internal priority.

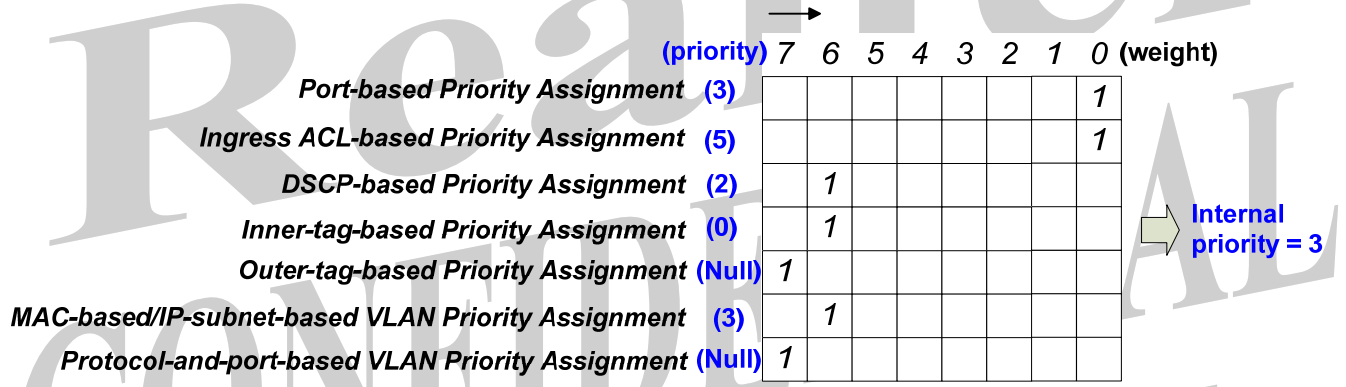


Figure 35: Priority Selection Weight Assignment Example

API REFERENCE

```
rtk_qos_priSelGroup_get(uint32 unit, uint32 grp_idx, rtk_qos_priSelWeight_t *pWeightOfPriSel);
rtk_qos_priSelGroup_set(uint32 unit, uint32 grp_idx, rtk_qos_priSelWeight_t *pWeightOfPriSel);
rtk_qos_portPriSelGroup_get(uint32 unit, rtk_port_t port, uint32 *pPriSelGrp_idx);
rtk_qos_portPriSelGroup_set(uint32 unit, rtk_port_t port, uint32 priSelGrp_idx);
```

PROGRAMMING EXAMPLE

Priority Selection Table Configuration

```
/* Priority selection table configuration */
pri_sel_tbl_idx = 2;

/* Set weight value to each priority assignment */
pri_sel_weight.weight_of_portBased = 0;
pri_sel_weightl.weight_of_inAcl = 0;
pri_sel_weight.weight_of_dscp = 6;
pri_sel_weight.weight_of_innerTag = 6;
pri_sel_weight.weight_of_outerTag = 7;
```

```
pri_sel_weight.weight_of_macVlan = 6;
pri_sel_weight.weight_of_protoVlan = 7;

/* Assign weight values to selection table */
rtk_qos_priSelGroup_set(unit, pri_sel_tbl_idx, &pri_sel_weight);

/* Set port 0 to use the configured priority selection table */
rtk_qos_portPriSelGroup_set(unit, 0, pri_sel_tbl_idx);
```

17.3 Internal Priority to QID Mapping

The device supports eight egress queues for each port (including CPU port). Queue number is configured by register QM_PORT_QNUM.

Table 125: QM_PORT_QNUM Register

Field Name	Bits	Description
QNUM	3	The number of queues for egress port. 3'b000: 1 output queue 3'b001: 2 output queues 3'b010: 3 output queues 3'b011: 4 output queues 3'b100: 5 output queues 3'b101: 6 output queues 3'b110: 7 output queues 3'b111: 8 output queues

Once the queue (traffic class) number is determined, it is used by the device to lookup below "Internal Priority to Queue ID Mapping Table" for picking up the output queue for a packet. Figure 36 is the suggested configuration values of QM_INTPRI2QID_CTRL register which is also the chip default value.

	Number of Traffic Class							
Internal Priority	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	1	1	1	1	2
3	0	0	0	1	1	2	2	3
4	0	1	1	2	2	3	3	4
5	0	1	1	2	3	4	4	5
6	0	1	2	3	4	5	5	6
7	0	1	2	3	4	5	6	7

Figure 36: Internal Priority to Queue ID Mapping Table

The internal priority mapping to egress queue (traffic class) can be modified by QM_INTPRI2QID_CTRL register.

Table 126: QM_INTPRI2QID_CTRL Register

Field Name	Bits	Description
INTPRI_QID	3	Internal priority mapping to egress queue ID

3'b000: queue ID 0 3'b001: queue ID 1 3'b010: queue ID 2 3'b011: queue ID 3 3'b100: queue ID 4 3'b101: queue ID 5 3'b110: queue ID 6 3'b111: queue ID 7

API REFERENCE

```
rtk_qos_queueNum_get(uint32 unit, uint32 *pQueue_num);
rtk_qos_queueNum_set(uint32 unit, uint32 queue_num);
rtk_qos_priMap_get(uint32 unit, uint32 queue_num, rtk_qos_pri2queue_t *pPri2qid);
rtk_qos_priMap_set(uint32 unit, uint32 queue_num, rtk_qos_pri2queue_t *pPri2qid);
```

PROGRAMMING EXAMPLE

Configure System Traffic Class Number and Internal Priority to QID Mapping

```
/* Traffic class configuration */
queue_num = 4;

/* Priority to QID mapping configurations */
pri2qid.pri2queue[0] = 0;
pri2qid.pri2queue[1] = 0;
pri2qid.pri2queue[2] = 1;
pri2qid.pri2queue[3] = 1;
pri2qid.pri2queue[4] = 2;
pri2qid.pri2queue[5] = 2;
pri2qid.pri2queue[6] = 3;
pri2qid.pri2queue[7] = 3;

/* Set weight value to each priority assignment */
rtk_qos_queueNum_set(unit, queue_num);

/* Configure internal priority to QID mapping */
rtk_qos_priMap_set(unit, queue_num, &pri2qid);
```

18 Egress Shaping

Packet scheduler is supported by each egress port. It is used to manage egress port bandwidth, egress queue (traffic class) bandwidth, and different scheduling algorithm WRR, WFQ, SP are supported. In traffic class bandwidth level, the device also supports fixed bandwidth feature to reserve certain bandwidth for a particular traffic class.

18.1 Scheduler Architecture

Egress bandwidth shaping is implemented in per-egress port scheduling module as below figure. The egress port rate is configured by EGR_RATE field in SCHED Table (refer to Table 127) and is implemented by leaky bucket (LB). Per egress port supports a SCHED Table configuration. For each traffic class queue, the scheduler supports two levels LB. One is the Average Peak Rate (APR) LB used to limit the maximum queue bandwidth. The other is the Weighted-Fair-Queue (WFQ) LB used to guarantee the minimum queue bandwidth.

The device supports two scheduling algorithms: WFQ which is byte-based and Weighted Round-Robin (WRR) which is packet-based. The algorithm is configured by SCHED_TYPE field in SCHED Table. Besides, the device provides WEIGHT_Qn configuration in SCHED Table to configure weighted value for each traffic class queue. In WFQ mode, system allots port rate (EGR_RATE) to each WFQ LB according to its weighted configuration.

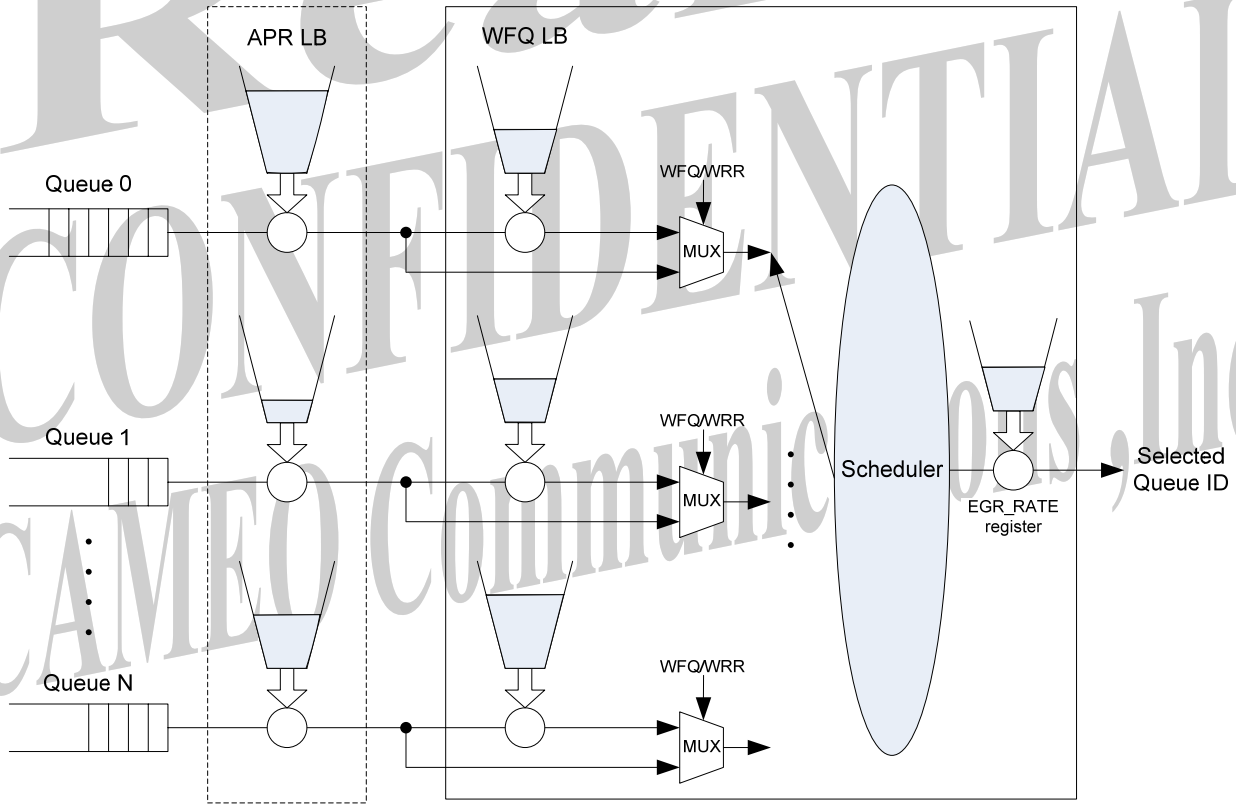


Figure 37: Egress Port Packet Scheduler

There are high and low thresholds for a leaky bucket. High threshold is checked while transmitting packets, and low threshold is checked while subtracts token. If token in LB is less than high threshold, packet can be transmitted. When a packet passes through, token equal to packet size is added into the bucket. At the mean time, LB token is subtracted by configured rate until the token is less than low threshold. The device supports global SCHED_LB_THR.APR_LB_SIZE register and SCHED_LB_THR.WFQ_LB_SIZE register in Table 129 to specify high thresholds of APR LB and WFQ LB. And the low thresholds of APR LB and WFQ LB are fixed to 0 and can't be

modified.

Besides, the device supports SCHED_CTRL.INC_IFG register in Table 128 to specify whether count inter-frame gap (IFG). Apart from packet size token is added to the LB, 20Bytes token is added to each LB when packet transmits if IFG is included.

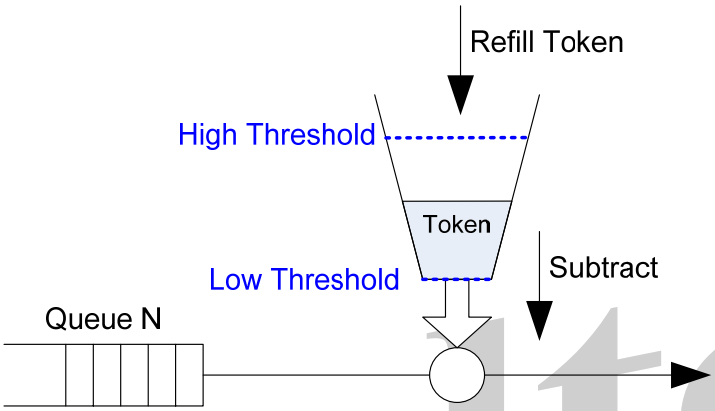


Figure 38: Leaky Bucket Architecture

Table 127: SCHED Table

Field Name	Bits	Description
SCHED_TYPE	1	Scheduler type selection for egress port. 1'b0: Weighted Fair Queue (WFQ) 1'b1: Weighted Round Robin (WRR)
WEIGHT_Qn	10	Weight value assign for WFQ/WRR for the egress queue n (n = 0~7). 10'h0: strict priority 10'h1 ~ 10'h3FF: weight value 1~1023
EGR_RATE	20	Egress bandwidthfor the egress port. (unit: 16Kbps) For Giga port, valid bit number is 16. 20'h0: blocking 20'h1 ~ 20'hF424: 16Kbps ~ 1Gbps 20'hFFFF: unlimited For 10G port, valid bit number is 20. 20'h0: blocking 20'h1 ~ 20'h98968: 16Kbps ~ 10Gbps 20'hFFFFF: unlimited
LB_APR_Qn	20	APR LB rate for the egress queue n (n = 0~7). (unit: 16Kbps) For Giga port, valid bit number is 16. 20'h0: blocking 20'h1 ~ 20'hF424: 16Kbps ~ 1Gbps 20'hFFFF: unlimited For 10G port, valid bit number is 20. 20'h0: blocking 20'h1 ~ 20'h98968: 16Kbps ~ 10Gbps 20'hFFFFF: unlimited
FIX_TKN_Qn	1	Fix bandwidth of the egress queue. If the feature is disabled, remaining WFQ tokens of the egress queue is shared to other egress queues in the same port. 1'b0: disable fixed bandwidth. 1'b1: enable fixed bandwidth. Doesn't share unused bandwidth to other queues.

Table 128: SCHED_CTRL Register

Field Name	Bits	Description
INC_IFG	1	Egress port bandwidth control includes/excludes the IFG (20Bytes). 1'b0: exclude 1'b1: include

Table 129: SCHED_LB_THR Register

Field Name	Bits	Description
APR_LB_SIZE	16	High threshold (bucket size) of APR LB in bytes for all ports. Unit: Byte
WFQ_LB_SIZE	16	High threshold (bucket size) of WFQ LB in bytes for all ports Unit: Byte

API REFERENCE

```
rtk_qos_schedulingAlgorithm_get(uint32 unit, rtk_port_t port, rtk_qos_scheduling_type_t
*pScheduling_type);
rtk_qos_schedulingAlgorithm_set(uint32 unit, rtk_port_t port, rtk_qos_scheduling_type_t scheduling_type);
rtk_rate_egrBandwidthCtrlIncludeIfg_get(uint32 unit, rtk_enable_t *pIfg_include);
rtk_rate_egrBandwidthCtrlIncludeIfg_set(uint32 unit, rtk_enable_t ifg_include);
```

18.2 Egress Port Bandwidth Management

The device supports egress bandwidth configuration for each egress port. Not only bandwidth configuration for egress port, but also weight configuration is supported for egress traffic class. In WRR mode, the ratio of transmitting packets number is controlled by the weight configuration. In WFQ mode, each queue can be allotted expected bandwidth by the weight configuration. For example, a port running WFQ with four egress queues Q0, Q1, Q2 and Q3 is configured to 500Mbps egress bandwidth. The weights of four queues are 1:2:3:4. When the packet injection rate is more than 500Mbps, the transmitting rates of Q0, Q1, Q2 and Q3 would be 50Mbps, 100Mbps, 150Mbps and 200Mbps, respectively.

API REFERENCE

```
rtk_qos_schedulingQueue_get(uint32 unit, rtk_port_t port, rtk_qos_queue_weights_t *pQweights);
rtk_qos_schedulingQueue_set(uint32 unit, rtk_port_t port, rtk_qos_queue_weights_t *pQweights);
rtk_rate_egrBandwidthCtrlEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);
rtk_rate_egrBandwidthCtrlEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
rtk_rate_egrBandwidthCtrlRate_get(uint32 unit, rtk_port_t port, uint32 *pRate);
rtk_rate_egrBandwidthCtrlRate_set(uint32 unit, rtk_port_t port, uint32 rate);
```

PROGRAMMING EXAMPLE

Configure Traffic Class Weight and Egress Bandwidth to the Specified Port

```
/* Queue weight configuration */
queue_weight.weights[0] = 1; /* queue 0 = weight 1 */
queue_weight.weights[1] = 2; /* queue 1 = weight 2 */
queue_weight.weights[2] = 3; /* queue 2 = weight 3 */
queue_weight.weights[3] = 4; /* queue 3 = weight 4 */
queue_weight.weights[4] = 5; /* queue 4 = weight 5 */
```

```
queue_weight.weights[5] = 6; /* queue 5 = weight 6 */
queue_weight.weights[6] = 0; /* queue 6 = strict priority */
queue_weight.weights[7] = 0; /* queue 7 = strict priority */

/* Bandwidth configuration */
rate = 100; /* unit = 16Kbps, rate = 100*16Kbps */

/* Set weight value to port 1 */
port = 1;
rtk_qos_schedulingQueue_set(unit, port, &queue_weight);

/* Set bandwidth configuration to port 1 */
rtk_rate_egrBandwidthCtrlEnable_set(unit, port, ENABLED);
rtk_rate_egrBandwidthCtrlRate_set(unit, port, rate);
```

18.3 Egress Port Bandwidth Management in CPU Port

Unlike normal port, CPU port supports both BPS and PPS mode for precisely control the bandwidth forwarding to CPU. The device provides SCHED_CTRL.LB_MODE_CPU_P register to specify bandwidth counting mode for CPU port. The configuration applies to both egress port rate and egress queue rate. In PPS mode, the meaning of egress port rate (EGR_RATE) and egress queue rate (LB_APR_Qn) are changed to Table 131.

Table 130: SCHED_CTRL Register

Field Name	Bits	Description
LB_MODE_CPU_P	1	Bandwidth process mode of egress port and egress queue to CPU port. 1'b0: BPS (byte per second) mode 1'b1: PPS (packet per second) mode

Table 131: SCHED Table for CPU Port in PPS mode

Field Name	Bits	Description
EGR_RATE	20	CPU port egress bandwidth in PPS mode. (unit: 1pps) Valid bit number is 16. 20'h0: blocking 20'h1 ~ 20'hFFFE: 1pps ~ 65534pps 20'hFFFF: unlimited
LB_APR_Qn	20	CPU port APR LB rate in PPS mode for the egress queue n (n=0~7). (unit: 1pps) Valid bit number is 16. 20'h0: blocking 20'h1 ~ 20'hFFFE: 1pps ~ 65534pps 20'hFFFF: unlimited

API REFERENCE

```
rtk_rate_egrBandwidthCtrlCPURateMode_get(uint32 unit, rtk_rate_rateMode_t *pRate_mode);
rtk_rate_egrBandwidthCtrlCPURateMode_set(uint32 unit, rtk_rate_rateMode_t rate_mode);
```

PROGRAMMING EXAMPLE

Configure Egress Bandwidth Counting Mode of CPU Port

```
/* Bandwidth control counting mode */
rate_mode = RATE_MODE_PKT; /* unit : packet */

/* Set egress bandwidth counting mode of CPU port */
rtk_rate_egrBandwidthCtrlCPURateMode_set(unit, rate_mode);
```

18.4 Egress Queue Bandwidth Management

In addition to egress port bandwidth configuration, the device also supports egress queue bandwidth configuration by LB_APR_Qn field in SCHED Table. The egress queue bandwidth configuration is implemented in APR LB.

Continue with same example described in section 18.2, now egress queue bandwidth of Q3 is set to 140Mbps. The transmitting rates of Q0, Q1, Q2 and Q3 will be 60Mbps, 120Mbps, 180Mbps and 140Mbps, respectively. However, if egress queue bandwidth of Q3 is configured over 200Mbps, the transmitting rates of 4 queues are keeping in 50Mbps, 100Mbps, 150Mbps and 200Mbps due to Q3 is limited by its weight value 4 in WFQ module.

API REFERENCE

```
rtk_rate_egrQueueBwCtrlEnable_get(uint32 unit, rtk_port_t port, rtk_qid_t queue, rtk_enable_t *pEnable);
rtk_rate_egrQueueBwCtrlEnable_set(uint32 unit, rtk_port_t port, rtk_qid_t queue, rtk_enable_t enable);
rtk_rate_egrQueueBwCtrlRate_get(uint32 unit, rtk_port_t port, rtk_qid_t queue, uint32 *pRate);
rtk_rate_egrQueueBwCtrlRate_set(uint32 unit, rtk_port_t port, rtk_qid_t queue, uint32 rate);
```

PROGRAMMING EXAMPLE

Configure Traffic Class Bandwidth to the Specified Port

```
/* Bandwidth configuration */
rate = 100; /* unit = 16Kbps, rate = 100*16Kbps */

/* Set bandwidth configuration to queue 7of port 1 */
rtk_rate_egrQueueBwCtrlEnable_set(unit, 1, 7, ENABLED);
rtk_rate_egrQueueBwCtrlRate_set(unit, 1, 7, rate);
```

18.5 Egress Queue Fixed Bandwidth Mechanism

The device supports reserved bandwidth function to each traffic class. The capability is used to retain private bandwidth for the particular traffic class, and it is configured by FIX_TKN_Qn field in SCHED Table. If a traffic class with FIX_TKN_Qn disabled has reset bandwidth, the reset bandwidth is distributed to other traffic classes. Take the same example described in section 18.2. No traffic is en-queued to Q3 and FIX_TKN_Q3 is disabled, the rates of Q0, Q1 and Q2 are nearly 83Mbps, 166Mbps and 249Mbps, respectively. Contrariwise, if the traffic class' FIX_TKN_Q3 is enabled, the unused bandwidth is not distributed to others. The rates of Q0, Q1, Q2 and Q3 would be 50Mbps, 100Mbps, 150Mbps and 0bps, respectively.

API REFERENCE

```
rtk_rate_egrQueueFixedBandwidthEnable_get(uint32 unit, rtk_port_t port, rtk_qid_t queue, rtk_enable_t
*pEnable);
rtk_rate_eqrQueueFixedBandwidthEnable_set(uint32 unit, rtk_port_t port, rtk_qid_t queue, rtk_enable_t
enable);
```

PROGRAMMING EXAMPLE

Configure Fix Bandwidth Function of the Particular Traffic Class

```
/* Configure fix bandwidth capability to queue 3 of port 1 */
rtk_rate_eqrQueueFixedBandwidthEnable_set(unit, 1, 3, ENABLED);
```

19 Egress Remarking

Before the packet is transmitted, the device supports flexible remarking function to remark the packet. The VLAN priority or DSCP value of a packet can be remarked to have different QoS process by next hop. The remarking types supported are listed below:

- Inner-tag priority
- Outer-tag priority
- IPv4 and IPv6 DSCP
- DEI flag

System per-egress-port provides configuration register RMK_PORT_RMK_EN_CTRL to enable the remark function for each QoS attributes.

Table 132: RMK_PORT_RMK_EN_CTRL Register

Field Name	Bits	Description
IPRI_RMK_EN	1	Enable inner priority remarking for a port. 1'b0: disable 1'b1: enable
OPRI_RMK_EN	1	Enable outer priority remarking for a port. 1'b0: disable 1'b1: enable
DSCP_RMK_EN	1	Enable DSCP remarking for a port. 1'b0: disable 1'b1: enable
DEI_RMK_EN	1	Enable DEI remarking for a port. 1'b0: disable 1'b1: enable

Note: The Egress ACL remarking action has higher priority than the remarking modules depicted in this chapter.

API REFERENCE

```
rtk_qos_1pRemarkEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);
rtk_qos_1pRemarkEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
rtk_qos_out1pRemarkEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);
rtk_qos_out1pRemarkEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
rtk_qos_dscpRemarkEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);
rtk_qos_dscpRemarkEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
rtk_qos_deiRemarkEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);
rtk_qos_deiRemarkEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
```

19.1 Inner-tag Priority Remarking

The inner-tag priority remarking module is not only used to remark inner-tag priority but also used to decide an inner-tag priority for inner-untag-in but inner-tag-out packet.

19.1.1 Inner-tag Priority Remarking Mechanism

System provides global two registers RMK_CTRL.IPRI_RMK_SRC and RMK_CTRL.IPRI_RMK_SRC_EXT to specify inner-tag priority remarking source. There are four inner-tag priority remarking sources: internal-priority, original inner-tag priority, original outer-tag priority and DSCP. Corresponding inner-tag remarking register RMK_IPRI_CTRL or RMK_DSCP2IPRI_CTRL is used for mapping new remarked priority according to the remark source. Internal-priority, original inner-tag priority and original outer-tag priority remarking share the configuration

register RMK_IPRI_CTRL. While DSCP remrking uses configuration register RMK_DSCP2IPRI_CTRL.

Table 133: RMK_CTRL Register

Field Name	Bits	Description
IPRI_RMK_SRC	1	Specify inner-priority remarking source. If original packet doesn't have inner-tag, system takes the register value IPRI_RMK_DFLT_PRI to be its inner-priority. 1'b0: internal priority 1'b1: original inner-priority
IPRI_RMK_SRC_EXT	2	Specify inner-priority remarking source. 2'b00: follow IPRI_RMK_SRC setting 2'b01: original outer-priority 2'b10: DSCP value 2'b11: reserved

Table 134: RMK_IPRI_CTRL Register

Field Name	Bits	Description
IPRIn	3	New inner-tag priority which is lookuped when remarking sources are internal-priority, original inner-priority, or original outer-priority. n=0~7.

Table 135: RMK_DSCP2IPRI_CTRL Register

Field Name	Bits	Description
DSCP2IPRIn	3	New inner-tag priority which is lookuped when remarking source is DSCP. n=0~63.

Table Index (IPRIn)	Remarked Value
Inner-priority 7/Outer-priority 7/Internal-priority 7	Inner-priority
Inner-priority 6/Outer-priority 6/Internal-priority 6	Inner-priority
Inner-priority 5/Outer-priority 5/Internal-priority 5	Inner-priority
Inner-priority 4/Outer-priority 4/Internal-priority 4	Inner-priority
Inner-priority 3/Outer-priority 3/Internal-priority 3	Inner-priority
Inner-priority 2/Outer-priority 2/Internal-priority 2	Inner-priority
Inner-priority 1/Outer-priority 1/Internal-priority 1	Inner-priority
Inner-priority 0/Outer-priority 0/Internal-priority 0	Inner-priority

Figure 39: Internal/Inner/Outer Priority to Inner-Tag Priority Remarking

Table Index (DSCP2IPRIn)	Remarked Value
DSCP 63	Inner-priority
DSCP 62	Inner-priority
……	Inner-priority
DSCP 2	Inner-priority
DSCP 1	Inner-priority
DSCP 0	Inner-priority

Figure 40: DSCP to Inner-Tag Priority Remarking

API REFERENCE

```
rtk_qos_1pRemarkSrcSel_get(uint32 unit, rtk_qos_1pRmkSrc_t *pType);
rtk_qos_1pRemarkSrcSel_set(uint32 unit, rtk_qos_1pRmkSrc_t type);
rtk_qos_1pRemark_get(uint32 unit, rtk_pri_t int_pri, rtk_pri_t *pDot1p_pri);
rtk_qos_1pRemark_set(uint32 unit, rtk_pri_t int_pri, rtk_pri_t dot1p_pri);
rtk_qos_dscp2Dot1pRemark_get(uint32 unit, uint32 dscp, rtk_pri_t *pDot1p_pri);
rtk_qos_dscp2Dot1pRemark_set(uint32 unit, uint32 dscp, rtk_pri_t dot1p_pri);
```

PROGRAMMING EXAMPLE

Inner-tag Priority Remarking Source and Remarking Value Configurations

```
/* Set DSCP to be inner-tag priority remarking source */
rmk_src = DOT_1P_RMK_SRC_DSCP;
rtk_qos_1pRemarkSrcSel_set(unit, rmk_src);

/* Configure DSCP to inner-tag priority remarking table value */
for (i=0; i<64; i++) {
    rtk_qos_dscp2Dot1pRemark_set(unit, i, 3);
}
```

19.1.2 Default Inner-tag Priority Assignment

The device provides register RMK_CTRL.IPRI_DFLT_SRC to specify inner-tag priority source for inner-untag-in but inner-tag-out packet. There are three prioritiy sources supported: from configuration register RMK_CTRL.IPRI_DFLT_PRI, copy ingress port-based priority, and copy internal-priority.

Table 136: RMK_CTRL Register

Field Name	Bits	Description
IPRI_DFLT_SRC	2	Inner-priority source for inner-untag-in but inner-tag-out packet. 2'b00: take IPRI_DFLT_PRI 2'b01: copy ingress port-based priority 2'b10: copy internal priority 2'b11: reserved
IPRI_DFLT_PRI	3	Deafult Inner-Tag priority.

RMK_CTRL.IPRI_DFLT_SRC configuration is also used for the case that an egress port enables inner-tag priority remarking and specify a particular remarking source, said DSCP, but the receiving packet is not an IP packet. The inner-tag priority for the egress packet is then determined by default inner-tag priority assignment.

API REFERENCE

```
rtk_qos_1pDfltPriSrcSel_get(uint32 unit, rtk_qos_1pDfltPriSrc_t *pType);
rtk_qos_1pDfltPriSrcSel_set(uint32 unit, rtk_qos_1pDfltPriSrc_t type);
rtk_qos_1pDfltPri_get(uint32 unit, rtk_pri_t *pDot1p_pri);
rtk_qos_1pDfltPri_set(uint32 unit, rtk_pri_t dot1p_pri);
```

19.2 Outer-tag Priority Remarking

The outer-tag priority remarking module is not only used to remark outer-tag priority but also used to decide an outer-tag priority for outer-untag-in but outer-tag-out packet.

19.2.1 Outer-tag Priority Remarking Mechanism

Outer-tag priority remarking is similar to inner-tag priority remarking. System provides global two registers RMK_CTRL.OPRI_RMK_SRC and RMK_CTRL.OPRI_RMK_SRC_EXT to specify outer-tag priority remarking source. There are four outer-tag priority remarking sources: internal-priority, original inner-tag priority, original outer-tag priority and DSCP. Corresponding outer-tag remarking register RMK_OPRI_CTRL or RMK_DSCP2OPRI_CTRL is used for mapping new remarked priority according to the remark source. Internal-priority, original inner-tag priority and original outer-tag priority remarking share the configuration register RMK_OPRI_CTRL. While DSCP remrking uses configuration register RMK_DSCP2OPRI_CTRL.

Table 137: RMK_CTRL Register

Field Name	Bits	Description
OPRI_RMK_SRC	1	Specify outer-priority remarking source. 1'b0: internal priority 1'b1: original outer-priority
OPRI_RMK_SRC_EXT	2	Specify outer-priority remarking source. 2'b00: follow OPRI_RMK_SRC setting 2'b01: original inner-priority 2'b10: DSCP value 2'b11: reserved

Table 138: RMK_OPRI_CTRL Register

Field Name	Bits	Description
OPRIn	3	New outer-tag priority which is lookuped when remarking sources are internal-priority, original inner-priority, or original outer-priority. n=0~7.

Table 139: RMK_DSCP2OPRI_CTRL Register

Field Name	Bits	Description
DSCP2OPRIn	3	New outer-tag priority which is lookuped when remarking source is DSCP. n=0~63.

Table Index (OPRIn)	Remarked Value
Inner-priority 7/Outer-priority 7/Internal-priority 7	Outer-priority
Inner-priority 6/Outer-priority 6/Internal-priority 6	Outer-priority
Inner-priority 5/Outer-priority 5/Internal-priority 5	Outer-priority
Inner-priority 4/Outer-priority 4/Internal-priority 4	Outer-priority
Inner-priority 3/Outer-priority 3/Internal-priority 3	Outer-priority
Inner-priority 2/Outer-priority 2/Internal-priority 2	Outer-priority
Inner-priority 1/Outer-priority 1/Internal-priority 1	Outer-priority
Inner-priority 0/Outer-priority 0/Internal-priority 0	Outer-priority

Figure 41: Internal/Inner/Outer Priority to Outer-Tag Priority Remarking

Table Index (DSCP2OPRIn)	Remarked Value
DSCP 63	Outer-priority
DSCP 62	Outer-priority
……	Outer-priority
DSCP 2	Outer-priority
DSCP 1	Outer-priority
DSCP 0	Outer-priority

Figure 42: DSCP to Outer-Tag Priority Remarking

API REFERENCE

```
rtk_qos_outer1pRemarkSrcSel_get(uint32 unit, rtk_qos_outer1pRmkSrc_t *pType);
rtk_qos_outer1pRemarkSrcSel_set(uint32 unit, rtk_qos_outer1pRmkSrc_t type);
rtk_qos_out1pRemark_get(uint32 unit, rtk_pri_t int_pri, rtk_pri_t *pDot1p_pri) ;
rtk_qos_out1pRemark_set(uint32 unit, rtk_pri_t int_pri, rtk_pri_t dot1p_pri) ;
rtk_qos_dscp2Outer1pRemark_get(uint32 unit, uint32 dscp, rtk_pri_t *pDot1p_pri);
rtk_qos_dscp2Outer1pRemark_set(uint32 unit, uint32 dscp, rtk_pri_t dot1p_pri);
```

PROGRAMMING EXAMPLE

Outer-tag Priority Remarking Source and Remarking Value Configurations

```
/* Set internal-priority to be outer-tag priority remarking source */
rmk_src = OUTER_1P_RMK_SRC_INT_PRI;
rtk_qos_outer1pRemarkSrcSel_set(unit, rmk_src);

/* Configure internal-priority to outer-tag priority remarking table value */
for (i=0; i<8; i++) {
      rtk_qos_out1pRemark _set(unit, i, 7);
}
```

19.2.2 Default Outer-tag Priority Assignment

Default outer-tag prioirity assignment is similar to inter-tag prioirity assignment except the configuration is per-port basis. Per port configuration which refer to ingress or egress port is determined by RMK_CTRL.OPRI_DFLT_CFG register. The device provides per port register RMK_PORT_OPRI_SRC_CTRL and RMK_PORT_OPRI_SRC_EXT_CTRL to specify outer-tag priority source for outer-untag-in but outer-tag-out packet. There are four prioritiy sources supported: from configuration register RMK_CTRL.OPRI_DFLT_PRI, copy ingress port-based priority, copy internal-priority, and copy inner-tag priority.

Similarly, RMK_PORT_OPRI_SRC_CTRL and RMK_PORT_OPRI_SRC_EXT_CTRL configurations are also used for the case that an egress port enables outer-tag priority remarking and specify a particular remarking source, said DSCP, but the receiving packet is not an IP packet. The outer-tag priority for the egress packet is then determined by default outer-tag priority assignment.

Table 140: RMK_CTRL Register

Field Name	Bits	Description
OPRI_DFLT_PRI	3	Outer-tag priority default value.
OPRI_DFLT_CFG	1	Specify configurations OPRI_DFLT_SRC and OPRI_DFLT_SRC_EXT are referred from ingress or egress port. 1'b0: ingress port 1'b1: egress port

Table 141: RMK_PORT_OPRI_SRC_CTRL Register

Field Name	Bits	Description
OPRI_DFLT_SRC	1	Specify default outer-priority source. 1'b0: copy from inner-priority. If ingress packet is an inner-untag packet, then ingress port-based priority is used. 1'b1: copy from internal priority

Table 142: RMK_PORT_OPRI_SRC_EXT_CTRL Register

Field Name	Bits	Description
OPRI_DFLT_SRC_EXT	2	Specify default outer-priority source. 2'b00: follow OPRI_DFLT_SRC setting 2'b01: use OPRI_DFLT_PRI 2'b10: copy ingress port-based priority 2'b11: reserved

API REFERENCE

```
rtk_qos_outer1pDfltPri_get(uint32 unit, rtk_pri_t *pDot1p_pri);
rtk_qos_outer1pDfltPri_set(uint32 unit, rtk_pri_t dot1p_pri);
rtk_qos_outer1pDfltPriCfgSrcSel_get(uint32 unit, rtk_qos_outer1pDfltCfgSrc_t *pDflt_sel);
rtk_qos_outer1pDfltPriCfgSrcSel_set(uint32 unit, rtk_qos_outer1pDfltCfgSrc_t dflt_sel);
rtk_qos_portOuter1pDfltPriSrcSel_get(uint32 unit, rtk_port_t port, rtk_qos_outer1pDfltSrc_t *pType);
rtk_qos_portOuter1pDfltPriSrcSel_set(uint32 unit, rtk_port_t port, rtk_qos_outer1pDfltSrc_t type);
```

PROGRAMMING EXAMPLE

Default Outer-tag Priority Source and Value Configurations

```
/* Set ingress port to be default outer-tag priority referring port */
dflt_port_sel = OUTER_1P_DFLT_CFG_SRC_INGRESS;
rtk_qos_outer1pDfltPriCfgSrcSel_set(unit, dflt_port_sel);

/* Configure default outer-tag priority value */
rtk_qos_outer1pDfltPri_set(unit, 3);
```

19.3 DSCP Remarking

DSCP (Differentiated Services Code Point) is a 6-bit value in the TOS field of IPv4 header and in the Traffic Class field of IPv6 header which is used to provide different Quality of Service for network traffic. Simlar to inner-tag and outer-tag priority remarking, DSCP remarking support configuration registers RMK_CTRL.DSCP_RMK_SRC and RMK_CTRL.DSCP_RMK_SRC_EXT to decide the remarking sources: internal-priority, original DSCP, original inner-priority, original outer-priority, and drop precedence (DP).

Table 143: RMK_CTRL Register

Field Name	Bits	Description
DSCP_RMK_SRC	2	Specify DSCP remarking source. 2'b00: internal priority 2'b01: original DSCP 2'b10: DP value 2'b11: reserved
DSCP_RMK_SRC_EXT	2	Specify DSCP remarking source. 2'b00: follow DSCP_RMK_SRC setting 2'b01: original inner-priority 2'b10: original outer-priority 2'b11: reserved

Table 144: RMK_DSCP_CTRL Register

Field Name	Bits	Description
DSCPn	6	New DSCP value. n=0~63.

The RMK_DSCP_CTRL is used to map a new DSCP value as below figure if DSCP remarking sources are internal-priority, original DSCP, original inner-priority, original outer-priority.

Table Index (DSCPn)	Remarked DSCP
DSCP 63	DSCP Value
DSCP 62	DSCP Value
……	……
DSCP 9	DSCP Value
DSCP 8	DSCP Value
DSCP 7/Internal-priority 7/Inner-priority 7/Outer-priority 7	DSCP Value
DSCP 6/Internal-priority 6/Inner-priority 6/Outer-priority 6	DSCP Value
DSCP 5/Internal-priority 5/Inner-priority 5/Outer-priority 5	DSCP Value
DSCP 4/Internal-priority 4/Inner-priority 4/Outer-priority 4	DSCP Value
DSCP 3/Internal-priority 3/Inner-priority 3/Outer-priority 3	DSCP Value
DSCP 2/Internal-priority 2/Inner-priority 2/Outer-priority 2	DSCP Value
DSCP 1/Internal-priority 1/Inner-priority 1/Outer-priority 1	DSCP Value
DSCP 0/Internal-priority 0/Inner-priority 0/Outer-priority 0	DSCP Value

Figure 43: Internal/DSCP/Inner/Outer Priority to DSCP Remarking

If the remarking source is specified to DP, the device remarks DSCP according to PHBs defined in RFC 2475 “DiffServ”:

- Default PHB(Best Effort): 000000(0)
- Expedited Forwarding PHB: 101110(46)
- Assured Forwarding PHB group(RFC 2597)
- Class Selector PHB: xxx000

Assured Forwarding (AF) Behavior Group				
	Class 1	Class 2	Class 3	Class 4
Low Drop (Green, DP=0)	AF11(001010)	AF21(010010)	AF31(011010)	AF41(100010)
Med Drop (Yellow, DP=1)	AF12(001100)	AF22(010100)	AF32(011100)	AF42(100100)
High Drop (Red, DP=2)	AF13(001110)	AF23(010110)	AF33(011110)	AF43(100110)

Figure 44: Assured Forwarding Behavior Group

Only AF PHBs utilize drop precedence (color) to forward/remark the packet. Therefore, system only remarks packet whose DSCP values are AF11~AF43. For example: a receiving packet with DSCP value AF11(001010) has been colored to red, then its DSCP value is remarked to AF13(001110). The remarking is done automatically and no configurations are needed. Packet with DSCP values other than AF11~AF43 is not remarked even the remarking source is specified to DP.

The DSCP remark both applies to IPv4 and IPv6 packet. The TOS field is remarked for IPv4 packet while the Traffic Classs field is remarked for IPv6 packet.

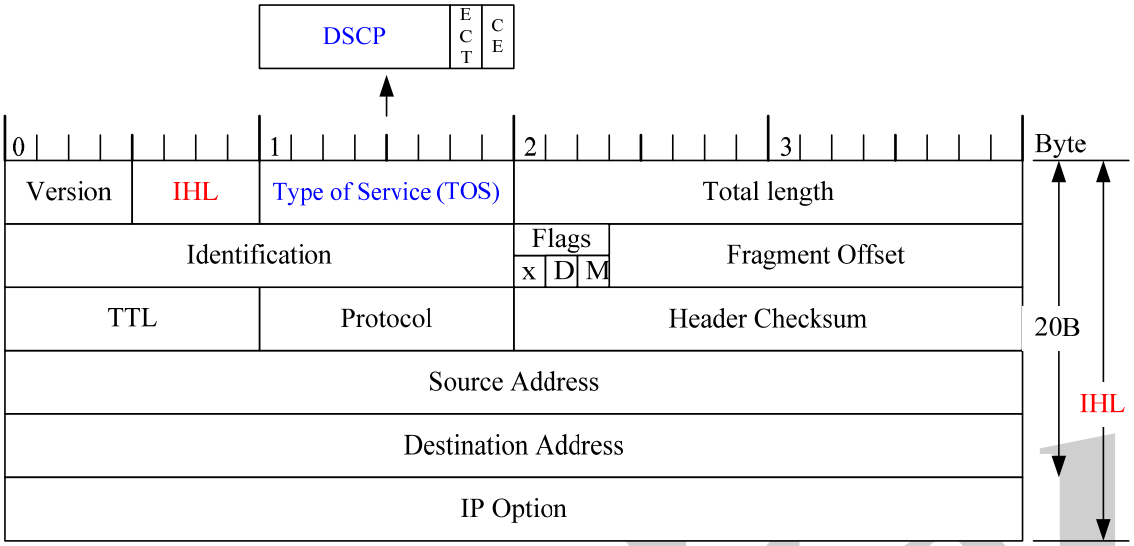


Figure 45: TOS field in IPv4 header format

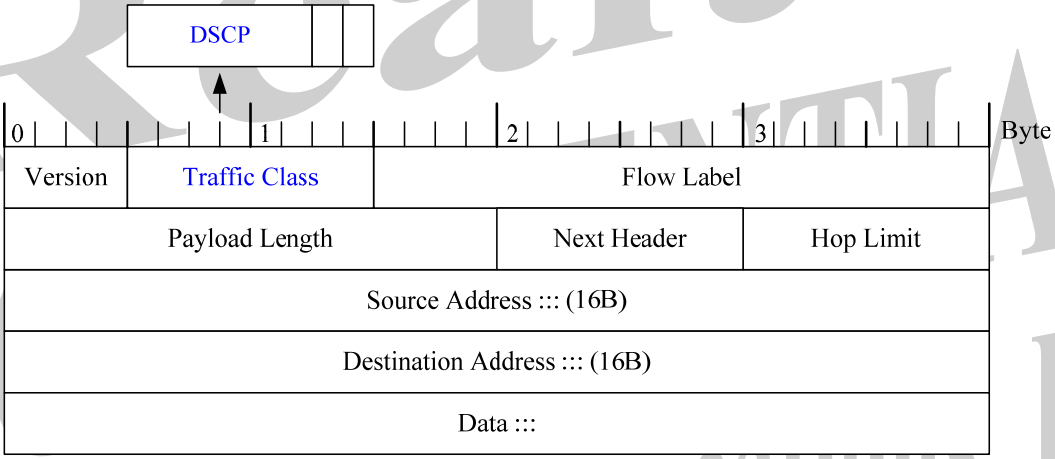


Figure 46: Traffic Class field in IPv6 header format

API REFERENCE

```
rtk_qos_dscpRemarkSrcSel_get(uint32 unit, rtk_qos_dscpRmkSrc_t *pType);
rtk_qos_dscpRemarkSrcSel_set(uint32 unit, rtk_qos_dscpRmkSrc_t type);
rtk_qos_dscpRemark_get(uint32 unit, rtk_pri_t int_pri, uint32 *pDscp);
rtk_qos_dscpRemark_set(uint32 unit, rtk_pri_t int_pri, uint32 dscp);
```

PROGRAMMING EXAMPLE

DSCP Remarking Remarking Source and DSCP Remarking Value Configuration

```
/* Set outer-tag to be tag target of DEI remarking for port 0 */
rmkSrc = DSCP_RMK_SRC_DSCP;
rtk_qos_dscpRemarkSrcSel_set(unit, rmkSrc);

/* Configure DSCP remarking table value */
for (i=0; i<64; i++) {
```

```
    rtk_qos_dscpRemark_set(unit, i, 63);
}
```

19.4 DEI Remarking

DEI (Drop Eligible Indicator) is a 1-bit flag in service VLAN tag which is used to indicate whether the packet has experienced congestion. The device supports flexible double tag manipulation and service VLAN tag can be reconginzed as inner tag or outer tag by configuration. Therefore, a per-egress-port register RMK_PORT_DEI_TAG_CTRL is provided to specify the target tag to be remarked. The DEI remarking is configured by RMK_DEI_CTRL register and DP is the only remarking source.

Table 145: RMK_PORT_DEI_TAG_CTRL Register

Field Name	Bits	Description
DEI_TAG_SEL	1	Specify inner tag or outer tag to remark DEI. 1'b0: inner tag 1'b1: outer tag

Table 146: RMK_DEI_CTRL Register

Field Name	Bits	Description
DP2DEI	3	DEI mapped from DP 0~2.

API REFERENCE

```
rtk_qos_portDEIRemarkTagSel_get(uint32 unit, rtk_port_t port, rtk_qos_deiSel_t *pType);
rtk_qos_portDEIRemarkTagSel_set(uint32 unit, rtk_port_t port, rtk_qos_deiSel_t type);
rtk_qos_deiRemark_get(uint32 unit, uint32 dp, uint32 *pDei);
rtk_qos_deiRemark_set(uint32 unit, uint32 dp, uint32 dei);
```

PROGRAMMING EXAMPLE

Tag Selection for DEI Remarking and DEI Remarking Value Configuration

```
/* Set outer-tag to be tag target of DEI remarking for port 0 */
tag_sel = DEI_SEL_OUTER_TAG;
rtk_qos_portDEIRemarkTagSel_set(unit, 0, tag_sel);

/* Configure DEI remarking table value */
rtk_qos_deiRemark_set(unit, 0, 0);
rtk_qos_deiRemark_set(unit, 1, 1);
rtk_qos_deiRemark_set(unit, 2, 1);
```

20 Meter Marker

The device supports meter mechanisms including dual leaky bucket, single rate three color marker (srTCM) and two rate three color marker (trTCM) which could be used for flow based rate limit or flow based packet coloring. Dual leaky bucket would drop packets if the traffic rate exceeds the limited rate while srTCM and trTCM would color packets to yellow or red. The packet's color maps to drop precedence (DP) value and affects its drop probability in Simplified Weighted Random Early Detection (S-WRED) module. Please refer to S-WRED chapter for the detail.

The flow characteristic to be metered is qualified by an ACL entry that specified "meter" action. The device supports 512 meter entries thus supports at most 512 flow based metering.

20.1 Meter Entry

The device supports 512 meter entries. The 512 meter entries are divided into 16 blocks thus each block contains 32 entries. Each entry is composed of TYPE, COLOR_AWARE, LB0_RATE, and LB1_RATE fields. TYPE decides the entry meter type which could be dual leaky bucket, srTCM, or trTCM. COLOR_AWARE only works for srTCM and trTCM types to indicate whether the packet's original color would participate the operation of new color. LB0_RATE and LB1_RATE are rate of two leaky buckets (however, srTCM only takes LB0_RATE since it is "single rate" marker). The unit is decided by METER_MODE (per block configuration) which could be byte based (granularity is 16Kbps) or packet based (granularity is 1 pps). The effective bit length of rates is 16 bits for byte based mode so the maximum rate is 1G bps. Effective length of rates is 20 bit for packet based mode so maximum rate is 1 million packets per second.

INCL_PREIFG register field indicates whether the rate counting of byte based mode should include or exclude preamble and IFG.

Table 147: Meter Entry Fields

Field Name	Bits	Description
TYPE	1	Meter Type 2'b00: Invalid meter entry 2'b01: DLB 2'b10: srTCM 2'b11: trTCM
COLOR_AWARE	1	Color aware or not. This bit takes effect when meter entry type is srTCM or trTCM. 1b'0: color unaware 1b'1: color aware
LB0_RATE	20	Rate of Leaky Bucket 0, the effective bit length is 16 bits for byte based mode and 20 bits for packet based mode.
LB1_RATE	20	Rate of Leaky Bucket 1, the effective bit length is 16 bits for byte based mode and 20 bits for packet based mode.

Table 148: METER_GLB_CTRL Register

Register Name	Field Name	Bits	Description
METER_GLB_CTRL	INCL_PREIFG	1	Packet length including preamble and IFG or not 0b0: exclude 0b1: include

Table 149: METER_MODE_CTRL Register

Register Name	Field Name	Bits	Description
METER_MODE_CTRL	METER_MODE	1	Mode of leaky bucket calculation unit of a meter block. 0b0: bps 0b1: pps

For a meter block, only meter entry with the lowest index would take effect if one packet hits multiple entries at the same block. Since there are 16 blocks, one packet can be affected by at most 16 meter entries concurrently. For example, if meter entry 0 limits ingress traffic of port 1 to 10 Mbps, meter entry 1 limits egress traffic of port 2 to 5 Mbps, and meter entry 40 limits VLAN 1 traffic to 20 Mbps. A VLAN 1 packet received from port 1 and transmitted to port 2 would be counted by meter entry 0 and meter entry 40. Meter entry 1 doesn't take effect because it is resided in the same block of meter entry 0.

Meter result could be "forward", "drop", "color green", "color yellow", or "color red". For a packet counted in multiple meter entries, the priority of meter result is "drop" > "color red" > "color yellow" > "color green" > "forward".

API REFERENCE

```
rtk_acl_meterMode_get(uint32 unit, uint32 idx, rtk_acl_meterMode_t *pMeterMode);
rtk_acl_meterMode_set(uint32 unit, uint32 idx, rtk_acl_meterMode_t meterMode);

rtk_acl_meterIncludeIfg_get(uint32 unit, rtk_enable_t *pIfg_include);
rtk_acl_meterIncludeIfg_set(uint32 unit, rtk_enable_t ifg_include);

rtk_acl_meterEntry_get(uint32 unit, uint32 meterIdx, rtk_acl_meterEntry_t *pMeterEntry);
rtk_acl_meterEntry_set(uint32 unit, uint32 meterIdx, rtk_acl_meterEntry_t *pMeterEntry);
```

20.2 Dual Leaky Bucket

Dual leaky bucket type meter contains two independent leaky buckets. When a packet hits the meter, by the length of packet, representative token would be added into the two leaky buckets. If the tokens in any one of the two leaky buckets exceed thresholds, the packet is dropped. The thresholds of dual leaky buckets are global configuration by BYTE_DLB_LB0_THR / BYTE_DLB_LB1_THR for byte based mode and PKT_DLB_LB0_THR / PKT_DLB_LB1_THR for packet based mode.

The rate of two leaky buckets is specified by register LB0_RATE and LB1_RATE. When the token size in a leaky bucket is greater than zero, the tokens would leak at speed of its rate configuration. If the traffic average rate is greater than the token leaky rate, the tokens would accumulate and exceed the threshold eventually thus the traffic would be rate limited.

One usage of the two leaky buckets is to set one leaky bucket with higher rate and lower threshold; another leaky bucket with lower rate and higher threshold. The lower rate bucket is used to limit the average rate, and the higher rate bucket is used to limit the burst rate. When a traffic burst-in with injection rate higher than high rate limit, it would exceed the threshold very quickly since the threshold is low; if the traffic rate is between high rate and low rate, traffic can be passed for a while till it reaches the threshold of low rate bucket.

Another common usage is to simply configured the rates and thresholds of two leaky buckets with the same values, so it works as a single leaky bucket rate policer.

Table 150: Dual Leaky Bucket Threshold Registers

Register Name	Field Name	Bits	Description
METER_BYTE_DLB_LB_THR_CTRL	BYTE_DLB_LB1_THR	16	Leaky bucket 1 threshold of dual leaky bucket type in unit of 128 bytes (Max. 8 M bytes). The configuration takes effect when METER_MODE is byte mode.
	BYTE_DLB_LB0_TH	16	Leaky bucket 0 threshold of dual leaky bucket type in

	R		unit of 128 bytes (max. 8 M bytes). The configuration takes effect when METER_MODE is byte mode.
METER_PKT_DLB_LB_THR_CTRL	PKT_DLB_LB1_THR	16	Leaky bucket 1 threshold of dual leaky bucket type in unit of 1 packet. The configuration takes effect when METER_MODE is packet mode.
	PKT_DLB_LB0_THR	16	Leaky bucket 0 threshold of dual leaky bucket type in unit of 1 packet. The configuration takes effect when METER_MODE is packet mode.

API REFERENCE

```
rtk_acl_meterBurstSize_get(uint32 unit, rtk_acl_meterMode_t meterMode, rtk_acl_meterBurstSize_t
*pBurstSize);
rtk_acl_meterBurstSize_set(uint32 unit, rtk_acl_meterMode_t meterMode, rtk_acl_meterBurstSize_t
*pBurstSize);

rtk_acl_meterEntry_get(uint32 unit, uint32 meterIdx, rtk_acl_meterEntry_t *pMeterEntry);
rtk_acl_meterEntry_set(uint32 unit, uint32 meterIdx, rtk_acl_meterEntry_t *pMeterEntry);
```

20.3 Single Rate Three Color Marker

The single rate three color marker type meter mechanism is based on RFC2697. LB0_RATE works as committed information rate (CIR) and LB1_RATE is ignored in srTCM. The threshold of LB0 works as committed burst size (CBS) and LB1 works as excess burst size (EBS). The thresholds are globally configurable for byte based and packet based mode. When a packet hits the meter, token equivalent to the packet would be added into either LB0 or LB1, and LB0 is the first choice whenever tokens are not greater than CBS. Tokens in LB0 would be subtracted at the rate of LB0_RATE. Only when LB0 is empty, tokens in LB1 would be subtracted also at the rate of LB0_RATE.

If COLOR_AWARE of the srTCM entry is disabled, the pre-color of the packet would be ignored. A packet is marked green if it doesn't exceed the CBS, yellow if it does exceed the CBS, but not the EBS, and red otherwise.

If COLOR_AWARE of the entry is enabled, the pre-color of packets would affect the color decision. The pre-color comes from drop precedence (DP) of the packet: drop precedence 0 maps to green color, value 1 maps to yellow color, and value 2 maps to red color. A packet is decided a DP value either from DEI or DSCP value, and configurable mapping tables of DEI-to-DP and DSCP-to-DP are provided (the packets without DEI nor DSCP values would be given DP 0). The color decision with pre-color involved is:

- if the packet has been pre-colored as green, and tokens in LB0 <= CBS, the packet is colored green and the token adds to LB0, else
- if the packet has been pre-colored as green or yellow, and LB1 <= EBS, the packet is colored yellow and the token adds to LB1, else
- the packet is colored red

Table 151: Drop Precedence Related Registers

Register Name	Field Name	Bits	Description
PRI_SEL_CTRL	DP_SRC	1	Select DP remapping source. 1'b0: DEI-based 1'b1: DSCP-based
PRI_SEL_DEI2DP_REMAP	DEI2DP_VAL	2	DEI to DP value mapping, value ranges from 0 to 2. The mapping table size is 2 for DEI 0 and DEI 1.
PRI_SEL_DSCP2DP_REMAP	DSCP2DP_VAL	2	DSCP to DP value mapping, value ranges from 0 to 2. The mapping table size is 64 for DSCP 0 to DSCP 63.

Table 152: srTCM Leaky Bucket Threshold Control Registers

Register Name	Field Name	Bits	Description

METER_BYTE_SRTCM_LB_THR_CTRL	BYTE_SRTCM_LB1_THR	16	Leaky bucket 1 threshold of single rate three color marker type in unit of 128 bytes (max. 8 M bytes). The configuration takes effect when METER_MODE is byte mode.
	BYTE_SRTCM_LB0_THR	16	Leaky bucket 0 threshold of single rate three color marker type in unit of 128 bytes (max. 8 M bytes). The configuration takes effect when METER_MODE is byte mode.
METER_PKT_SRTCM_LB_THR_CTRL	PKT_SRTCM_LB1_THR	16	Leaky bucket 1 threshold of single rate three color marker type in unit of 1 packet. The configuration takes effect when METER_MODE is packet mode.
	PKT_SRTCM_LB0_THR	16	Leaky bucket 0 threshold of single rate three color marker type in unit of 1 packet. The configuration takes effect when METER_MODE is packet mode.

API REFERENCE

```
rtk_qos_dpSrcSel_get(uint32 unit, rtk_qos_dpSrc_t *pType);
rtk_qos_dpSrcSel_set(uint32 unit, rtk_qos_dpSrc_t type);
rtk_qos_deiDpRemap_get(uint32 unit, uint32 dei, uint32 *pDp) ;
rtk_qos_deiDpRemap_set(uint32 unit, uint32 dei, uint32 dp) ;
rtk_qos_dscpDpRemap_get(uint32 unit, uint32 dscp, uint32 *pDp) ;
rtk_qos_dscpDpRemap_set(uint32 unit, uint32 dscp, uint32 dp) ;

rtk_acl_meterBurstSize_get(uint32 unit, rtk_acl_meterMode_t meterMode, rtk_acl_meterBurstSize_t
*pBurstSize);
rtk_acl_meterBurstSize_set(uint32 unit, rtk_acl_meterMode_t meterMode, rtk_acl_meterBurstSize_t
*pBurstSize);

rtk_acl_meterEntry_get(uint32 unit, uint32 meterIdx, rtk_acl_meterEntry_t *pMeterEntry);
rtk_acl_meterEntry_set(uint32 unit, uint32 meterIdx, rtk_acl_meterEntry_t *pMeterEntry);
```

20.4 Two Rate Three Color Marker

The two rate three color marker type meter mechanism is based on RFC2698. LB0_RATE works as committed information rate (CIR) and associated threshold is committed burst size (CBS). LB1_RATE works as peak information rate (PIR) and the associated threshold is peak burst size (PBS). The thresholds for trTCM meter entries are global configurable for byte based and packet based mode. When a packet hits the meter, token equivalent to the packet would be added to LB0 and LB1 whenever the tokens in LB0 and LB1 do not exceed its threshold. Tokens in LB0 would be subtracted at the rate of LB0_RATE till LB0 is empty; tokens in LB1 would be subtracted at the rate of LB1_RATE till it's empty. For normal usage, LB0_RATE should be configured less than LB1_RATE since LB1_RATE is peak rate that reflects the red color.

In trTCM color blind mode, the pre-color of the packet would be ignored. A packet is marked red if it exceeds the PBS (threshold of LB1), yellow if it exceeds the CBS (threshold of LB0), and green otherwise.

In trTCM color aware mode, the pre-color of packets would affect the color decision. The color decision with pre-color involved is:

- if the packet has been pre-colored as red, or tokens in LB1 > PBS, the packet is colored red, else
- if the packet has been pre-colored as yellow, or LB0 > CBS, the packet is colored yellow and the token adds to LB1, else
- the packet is colored green, the token adds to LB0 and LB1

Table 153: trTCM Leaky Bucket Threshold Control Registers

Register Name	Field Name	Bits	Description
METER_BYTE_TRTCM_LB_THR_CTRL	BYTE_TRTCM_LB1_THR	16	Leaky bucket 1 threshold of two rate three color marker type in unit of 128 bytes (max. 8Mbytes). The configuration takes effect when METER_MODE is byte mode.
	BYTE_TRTCM_LB0_THR	16	Leaky bucket 0 threshold of two rate three color marker type in unit of 128 bytes (max/ 8Mbytes). The configuration takes effect when METER_MODE is byte mode.
METER_PKT_TRTCM_LB_THR_CTRL	PKT_TRTCM_LB1_THR	16	Leaky bucket 1 threshold of two rate three color marker type in unit of 1 packet. The configuration takes effect when METER_MODE is packet mode.
	PKT_TRTCM_LB0_THR	16	Leaky bucket 0 threshold of two rate three color marker type in unit of 1 packet. The configuration takes effect when METER_MODE is packet mode.

API REFERENCE

*rtk_qos_dpSrcSel_get(uint32 unit, rtk_qos_dpSrc_t *pType);*
rtk_qos_dpSrcSel_set(uint32 unit, rtk_qos_dpSrc_t type);
*rtk_qos_deiDpRemap_get(uint32 unit, uint32 dei, uint32 *pDp) ;*
rtk_qos_deiDpRemap_set(uint32 unit, uint32 dei, uint32 dp) ;
*rtk_qos_dscpDpRemap_get(uint32 unit, uint32 dscp, uint32 *pDp) ;*
rtk_qos_dscpDpRemap_set(uint32 unit, uint32 dscp, uint32 dp) ;

*rtk_acl_meterBurstSize_get(uint32 unit, rtk_acl_meterMode_t meterMode, rtk_acl_meterBurstSize_t *pBurstSize);*
*rtk_acl_meterBurstSize_set(uint32 unit, rtk_acl_meterMode_t meterMode, rtk_acl_meterBurstSize_t *pBurstSize);*

*rtk_acl_meterEntry_get(uint32 unit, uint32 meterIdx, rtk_acl_meterEntry_t *pMeterEntry);*
*rtk_acl_meterEntry_set(uint32 unit, uint32 meterIdx, rtk_acl_meterEntry_t *pMeterEntry);*

20.5 Exceed Flag

For each meter entry, the device provides exceed flag LB_EXCEED to record if any packet was dropped (if entry is dual leaky bucket type) or colored red (if entry is srTCM or trTCM). There are total 512 bits exceed flag for 512 meter entries. 512 exceed flags are resided in 16 registers. In order to reduce the flag retrieving time, the device provides aggregated exceed flag LB_GLB_EXCEED that aggregates every 16 exceed flags to 1 bit. For 512 meter entries, there are 32 bits aggregated exceed flag. If one or more exceed flags of the 16 aggregated entries are on, the aggregated exceed flag would be on. For example, if only bit 3 (index starts from bit 0) of aggregated exceed flag is on, it means one or more entries in meter entry 48 to 63 hit exceed flag.

Table 154: Exceed Flag Registers

Register Name	Field Name	Bits	Description
METER_LB_EXCEED_STS	LB_EXCEED	1	1 bit flag per meter entry to show if the meter entry ever drops a packet or color a packet to red. Write 1 to clear. 0b0: no packet is dropped or colored red 0b1: packets are ever dropped or colored red
METER_LB_GLB_EXCEED_STS	LB_GLB_EXCEED	1	Aggregated exceeding status. A bit stands for 16 bits aggregated result of LB_EXCEED, for example, LB_GLB_EXCEED[0] aggregates LB_EXCEED[0][0] LB_EXCEED[0][15]. If any of the cared 16 bits of LB_EXCEED is 1, the respective LB_GLB_EXCEED bit is 1. The register is

read only.

API REFERENCE

```
rtk_acl_meterExceed_get(uint32 unit, uint32 meterIdx, uint32 *pIsExceed);
rtk_acl_meterExceedAggregation_get(uint32 unit, uint32 *pExceedMask);
```

21 Simplified Weighted Random Early Detection

In general network devices, it sends the pause frame or back pressure when flow control is enabled to slow down the transmitting speed of link partner when its resource is insufficient. When flow control is disabled, packet is dropped if the device's resource is insufficient. However, the drop would cause deterioration of TCP performance.

The device doesn't only support egress drop mechanism but also Simplified Weighted Random Early Detection (SWRED) are provided for the condition of flow control disabled. The drop algorithm is selected by WRED_GLB_CTRL register. SWRED can improve TCP performance by randomly early drop packet in advance. In addition, SWRED statistically drops more packets when system congestion becomes worser. Therefore, traffic sources that generate the most traffic are more likely to be slowed down than traffic sources that generate little traffic. SWRED provides individual thresholds and weights for different drop precedence, allowing you to provide different qualities of service for different traffic.

Table 155: WRED_GLB_CTRL Register

Field Name	Bits	Description
DROP_TYPE_SEL	1	Select egress drop algorithm (CPU port doesn't support SWRED). 1'b0: Egress Tail Drop 1'b1: Simplified Weighted Random Early Detection (SWRED)

API REFERENCE

```
rtk_qos_congAvoidAlgo_get(uint32 unit, rtk_qos_congAvoidAlgo_t *pAlgo);
rtk_qos_congAvoidAlgo_set(uint32 unit, rtk_qos_congAvoidAlgo_t algo);
```

21.1 Egress Port and Queue SWRED Drop

The device supports three sets of threshold configurations for drop precedence 0~2 to provide different drop criteria. For each received packet, it is given a Drop Precedence value. Generally, the red color packet should have higher drop probability than yellow and green and this is archieved by setting proper thresholds and drop probaility.

For egress port drop, the device per Drop Precedence supports a global configuration registers DROP_RATE_DP, THMAX_DP and THMIN_DP. The configuration registers are applied to all egress ports.

Table 156: WRED_PORT_THR_CTRL Regiser

Field Name	Bits	Description
DROP_RATE_DP	8	Drop rate (D_{rate}) of drop precedence *d* (*d*=0~2) for all ports. Maximum is 255.
THMAX_DP	12	Maximum drop threshold of drop precedence *d* (*d*=0~2) for all ports. Unit: page, 1 page = 360Byte
THMIN_DP	12	Minimum drop threshold of drop precedence *d* (*d*=0~2) for all ports. Unit: page, 1 page = 360Byte

For egress queue drop, the device supports global configuration registers Q_DROP_RATE_DP, Q_THMAX_DP and Q_THMIN_DP for each queue.

Table 157: WRED_QUEUE_THR_CTRL Register

Field Name	Bits	Description
Q_DROP_RATE_DP	8	Drop rate (D_{rate}) of drop precedence *d* (*d*=0~2) for egress queue n

		(n=0~7). Maximum is 255.
Q_THMAX_DP	12	Maximum drop threshold of drop precedence *d* (*d*=0~2) for egress queue n (n=0~7). Unit: page, 1 page = 360Byte
Q_THMIN_DP	12	Minimum drop threshold of drop precedence *d* (*d*=0~2) for egress queue n (n=0~7). Unit: page, 1 page = 360Byte

API REFERENCE

```
rtk_qos_congAvoidSysThresh_get(uint32 unit, uint32 dp, rtk_qos_congAvoidThresh_t
*pCongAvoidThresh);
rtk_qos_congAvoidSysThresh_set(uint32 unit, uint32 dp, rtk_qos_congAvoidThresh_t
*pCongAvoidThresh);
rtk_qos_congAvoidSysDropProbability_get(uint32 unit, uint32 dp, uint32 *pProbability);
rtk_qos_congAvoidSysDropProbability_set(uint32 unit, uint32 dp, uint32 probability)
rtk_qos_congAvoidGlobalQueueThresh_get(uint32 unit, rtk_qid_t queue, uint32 dp,
rtk_qos_congAvoidThresh_t *pCongAvoidThresh);
rtk_qos_congAvoidGlobalQueueThresh_set(uint32 unit, rtk_qid_t queue, uint32 dp,
rtk_qos_congAvoidThresh_t *pCongAvoidThresh);
rtk_qos_congAvoidGlobalQueueDropProbability_get(uint32 unit, rtk_qid_t queue, uint32 dp, uint32
*pProbability);
rtk_qos_congAvoidGlobalQueueDropProbability_set(uint32 unit, rtk_qid_t queue, uint32 dp, uint32
probability);
```

PROGRAMMING EXAMPLE

SWRED Port-based Threshold and Drop Rate Configurations

```
/* Configure drop rate 64 to drop precedence 1 of all ports */
rtk_qos_congAvoidSysDropProbability_set(unit, 1, 64);
;
/* Configure SWRED maximum and minimum thresholds to drop precedence 1 of all ports */
congAvoidThresh.maxThresh = 400;
congAvoidThresh.minThresh = 100;
rtk_qos_congAvoidSysThresh_set(unit, 1, &congAvoidThresh);
```

21.2 Architecture Overview

SWRED module checks below criteria before the packets are en-queued:

- Flow control ability of the packet's ingress port is disabled.
- Currently total queue length of egress port (K_{port}) is greater than minimum port-based threshold or currently egress queue length (K_{queue}) is greater than minimum queue-based threshold.
- System resource is in congestion state.

If the three criteria are all met, the packet has the packet-drop probability (*P*) to be dropped.

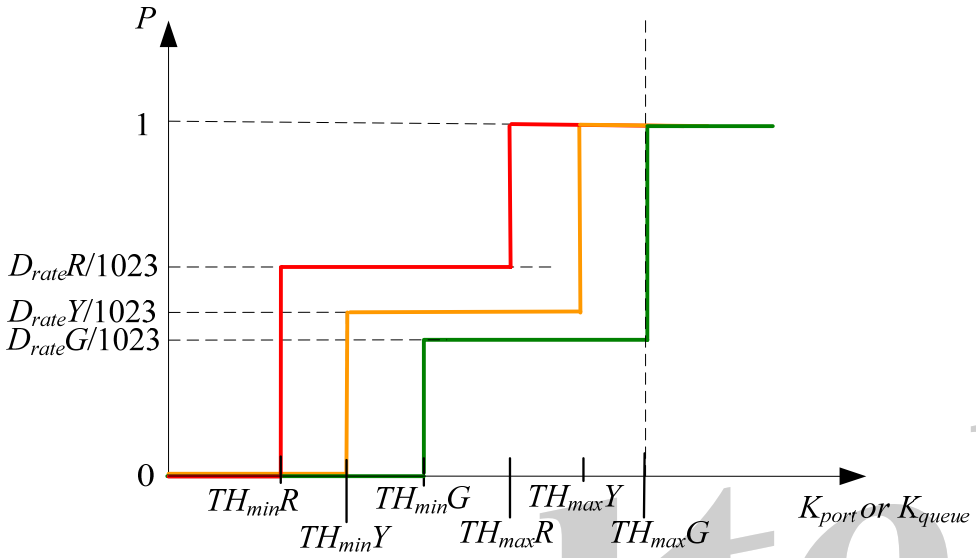


Figure 47: SWRED Parameter Reference Setting For Different Precedence

The device calculates parameters K_{port} and K_{queue} every system clock cycle for port-based and queue-based. Packet-drop probabilities of port-based P_{port} and queue-based P_{queue} are also updated every system clock cycle by following formulas. The SWRED flow chart is shown in Figure 48: SWRED Flow Chart.

$$P_{port} = 0, \quad if \quad K_{port} < \text{THMIN\_DP} \quad \text{.....(a)}$$

$$P_{port} = 1, \quad if \quad K_{port} \geq \text{THMAX\_DP} \quad \text{.....(b)}$$

$$P_{port} = \frac{\text{D\_RATE\_DP}}{1023}, \quad if \quad \text{THMIN\_DP} \leq K_{port} < \text{THMAX\_DP} \quad \text{.....(c)}$$

$$P_{queue} = 0, \quad if \quad K_{queue} < \text{Q\_THMIN\_DP} \quad \text{.....(d)}$$

$$P_{queue} = 1, \quad if \quad K_{queue} \geq \text{Q\_THMAX\_DP} \quad \text{.....(e)}$$

$$P_{queue} = \frac{\text{Q\_D\_RATE\_DP}}{1023}, \quad if \quad \text{Q\_THMIN\_DP} \leq K_{queue} < \text{Q\_THMAX\_DP} \quad \text{.....(f)}$$

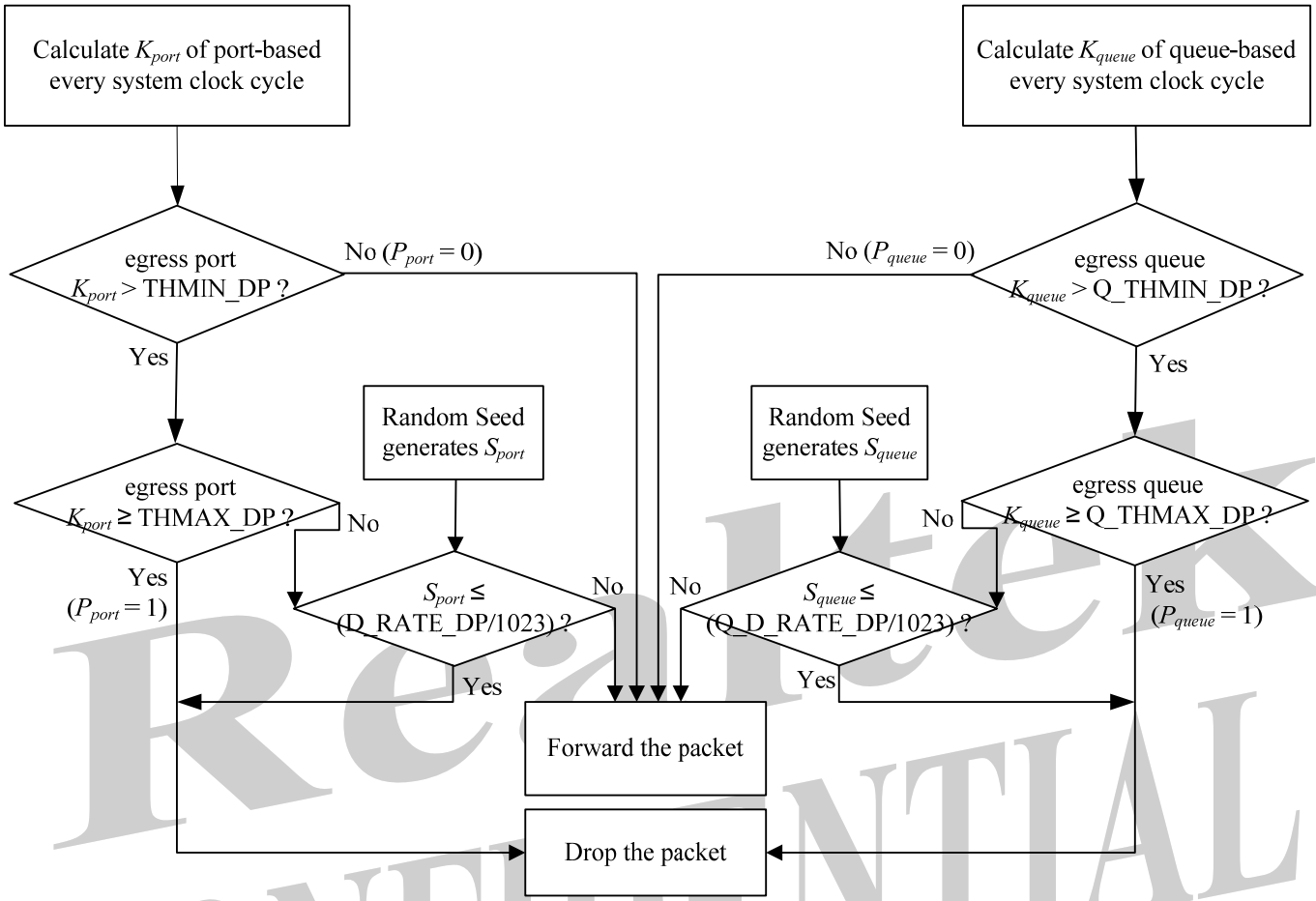


Figure 48: SWRED Flow Chart

22 NIC (Network Interface Controller)

NIC connected with CPU MAC (MAC 52) is used for receiving/transmitting packets from/to CPU. NIC is a DMA engine which is in charge of moving packet between CPU (Memory) and Switch Core.

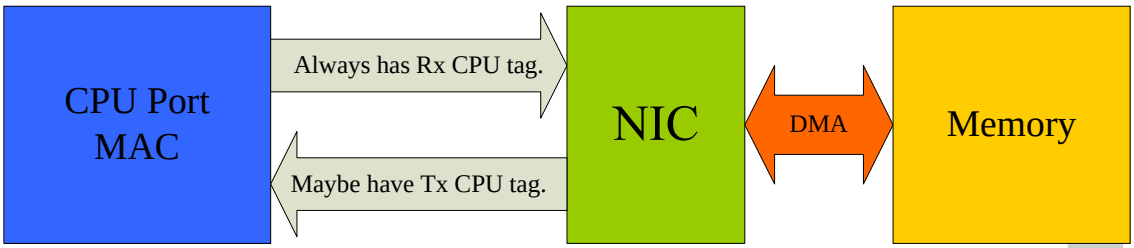


Figure 49: NIC and CPU MAC functional blocks

22.1 Packet Descriptor

The descriptor (drv_nic_pkt_t) below in SDK NIC driver is used to describe a packet:

```
FilePath: system/include/drv/nic/common.h

typedef struct drv_nic_pkt_s
{
    uint8   *head;          /* pointer to the head of the packet data buffer */
    uint8   *data;          /* pointer to the base address of the packet */
    uint8   *tail;          /* pointer to the end address of the packet */
    uint8   *end;           /* pointer to the end of the packet data buffer */
    uint32 length;          /* packet length when the packet is valid (not a empty data buffer) */
    void    *buf_id;        /* pointer to the user-defined packet descriptor */

    uint8   as_txtag;       /* 0: without tx-tag, 1: with tx-tag */
    union
    {
        /* Reception information */
        struct
        {
            ... (ignored, refer to CPU tag) ...
        } rx_tag;

        /* Transmission information */
        struct
        {
            ... (ignored, refer to CPU tag) ...
        } tx_tag;
    };

    struct drv_nic_pkt_s *next;             /* pointer to next packet struct if exists */
} drv_nic_pkt_t;
```

Below figure demonstrates how a jumbo packet is described:

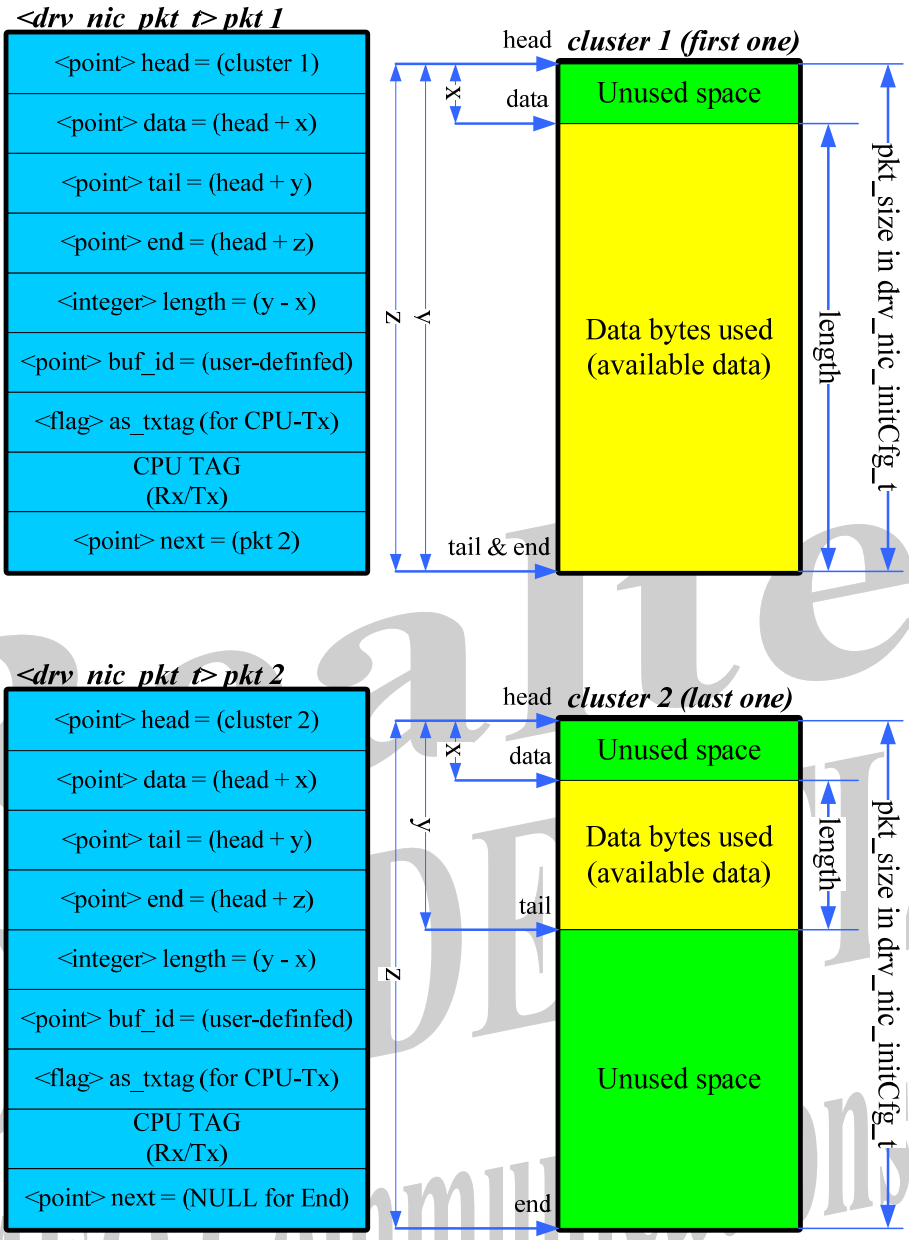


Figure 50: Example of Multiple Clusters Packet

22.2 Packet Reception (CPU-Rx)

The device provides 8 Rx rings mapped to the eight egress queue of the CPU MAC. When the packet is forwarded to CPU, the NIC performs a DMA Write to move the packet from the CPU MAC to the CPU memory.

API REFERENCE

FilePath: system/include/drv/nic/common.h

```
typedef enum drv_nic_rx_e
{
    NIC_RX_NOT_HANDLED = 0,     /* The packet is not handled, continue processing */
    NIC_RX_HANDLED,             /* The packet is handled and caller takes control of packet data */
    NIC_RX_HANDLED_OWNED,       /* The packet is handled and processing is completed */
} drv_nic_rx_t;

typedef int32 (*drv_nic_pkt_alloc_f)(uint32 unit, int32 size, uint32 flags, drv_nic_pkt_t **ppPacket);
typedef int32 (*drv_nic_pkt_free_f)(uint32 unit, drv_nic_pkt_t *pPacket);
typedef drv_nic_rx_t (*drv_nic_rx_cb_f)(uint32 unit, drv_nic_pkt_t *pPacket, void *pCookie);

typedef struct drv_nic_initCfg_s
{
    int32                   pkt_size;       /* Maximum bytes to support in a packet */
    drv_nic_pkt_alloc_f     pkt_alloc;      /* Packet structure and packet data allocation routine */
    drv_nic_pkt_free_f      pkt_free;       /* Packet structure and packet data deallocation routine */
} drv_nic_initCfg_t;

drv_nic_init(uint32 unit, drv_nic_initCfg_t *pInitCfg);
drv_nic_rx_register(uint32 unit, uint8 priority, drv_nic_rx_cb_f fRxCb, void *pCookie, uint32 flags);

drv_nic_rx_unregister(uint32 unit, uint8 priority, drv_nic_rx_cb_f fRxCb);
drv_nic_rx_start(uint32 unit);
drv_nic_rx_stop(uint32 unit);
```

During NIC driver initialization '*drv_nic_init*', it calls the user-defined function `*pkt_alloc*` in `*drv_nic_initCfg_t*` with `*pkt_size*` to prepare the buffer called Cluster to receive packet.

When a packet has been written to the Cluster (Memory), NIC notifies CPU to receive the packet with an interrupt signal (RX_DONE). Then the NIC driver callback the Rx (handler) callback function `*fRxCb*` that registered with `drv_nic_rx_register` API function according to priority. The parameter `*pCookie*` returned as an input parameter and user can use it to pass the application data. The parameter `*flags*` is optional which is reserved for future use.

The Rx handler can return one of the following values to indicate the result of handling.

1. NIC_RX_NOT_HANDLED – the packet is not handled. This return value indicates NIC driver to invoke the next Rx handler.
2. NIC_RX_HANDLED – the packet is handled by the Rx handler. This return value indicates the packet should be freed by the caller (NIC driver) by invoking the `*pkt_free*` in `*drv_nic_initCfg_t*`.
3. NIC_RX_HANDLED_OWNED – the packet is handled by the Rx handler. This return value indicates Caller (NIC driver) should not free the packet, and the packet free will be done by user.

Developer can invoke `*drv_nic_rx_start`/`drv_nic_rx_stop*` API to start/stop the Rx DMA process.

Table 158: DMA_IF_CTRL.RX Register

Field Name	Bits	Description
RX_EN	1	Enable Rx DMA process 1'b0: disable 1'b1: enable

According to the introduction above, the basic procedures to receive packets should be:

1. Invoke *drv_nic_init* API to initialize NIC if it has not been invoked yet.
2. Invoke *drv_nic_rx_register* API to register a Rx packet handler.
3. Invoke *drv_nic_rx_start* API to enable the Rx DMA process.

The Rx packet handler can retrieve related information of the received packet with the CPU Rx-Tag. Please refer to the CPU Tag Developer Guide for the detail.

22.3 Packet Transmission (CPU-Tx)

The device provides 2 Tx rings (high/low priority) to transmit packet through CPU MAC to front ports.When CPU transmits a packet, the NIC performs a DMA Read to move the packet from CPU memory to CPU MAC.

API REFERENCE

```
typedef void (*drv_nic_tx_cb_f)(uint32 unit, drv_nic_pkt_t *pPacket, void *pCookie);

drv_nic_init(uint32 unit, drv_nic_initCfg_t *pInitCfg);
drv_nic_pkt_alloc(uint32 unit, int32 size, uint32 flags, drv_nic_pkt_t **ppPacket);
drv_nic_pkt_free(uint32 unit, drv_nic_pkt_t *pPacket);
drv_nic_pkt_tx(uint32 unit, drv_nic_pkt_t *pPacket, drv_nic_tx_cb_f fTxCb, void *pCookie);
```

The SDK driver select the Tx queue according to the priority in CPU-Tx tag, it use the high queue if the priority value (0~7) is over 3.

CPU can invoke API `*drv_nic_pkt_tx*` to transmit a packet. If the caller want to be notified after packet is transmitted successfully, it can use the parameter `*fTxCb*` - Tx callback function. The NIC driver invokes the `*fTxCb*` function with `*pCookie*` (application data returned with callback, it can be null) parameter to notify the caller if `*fTxCb*` is not NULL, then the packet should be freed by user. If `*fTxCb*` is NULL, the NIC driver automatically invokes the `*pkt_free*` callback function to free the packet.

Table 159: DMA_IF_CTRL.TX Register

Field Name	Bits	Description
TX_EN	1	Enable Tx DMA process 1'b0: disable 1'b1: enable
TX_FETCH	1	Tx descriptor fetch notify (Set this bit to trigger the packet send) Note: Auto clear to 0 after trigger.

The basic procedures to transmit packet should be:

1. Invoke *drv_nic_init* API to initialize NIC if it has not been invoked yet.
2. Invoke *drv_nic_pkt_alloc* API to allocate an empty packet buffer for use.
3. Write raw data into the packet buffer (cluster).
4. Invoke *drv_nic_pkt_tx* API to send this packet.

23 CPU Tag

CPU Tag is a 12-Byte payload which is inserted between Source MAC Address and EtherType field. CPU Tag has different meaning for CPU Rx and Tx. It is used to indicate the packet processing information when CPU Rx and used to instruct the switch processing behavior when CPU Tx.

CPU Rx Tag is inserted by CPU MAC when the switch forwards packet to CPU. CPU Tx Tag is inserted by NIC when CPU transmits packet to switch and it is removed before transmitting to front ports.

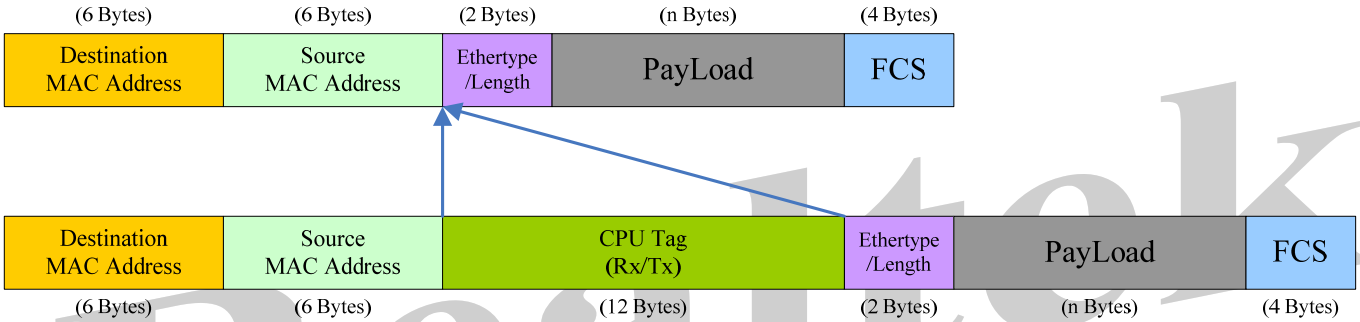


Figure 51: CPU Tag Position

23.1 CPU Rx Tag

The CPU Rx Tag provides information regarding the receiving packet and its payload is shown as below:

Table 160: Field Description of CPU Rx Tag

Field Name	Bits	Description
SPN	5	Ingress port which the packet came from. (0~51)
MIR_HIT	4	Mirroring hit status for 4-sets of mirror configuration. 4`b[3:0] = { set 3, set 2, set 1, set 0 }
ACL_IDX	12	Indicate the hit ACL entry index. Valid if ACL_HIT=1.
ACL_HIT	1	Indicate whether ACL hit.
OTAGIF	1	Indicate whether the packet contains outer tag.
ITAGIF	1	Indicate whether the packet contains inner tag.
FORWARD_VID	12	Forwarding VLAN ID
QID	3	Indicate the egress queue the packet came from.
ATK_TYPE	5	Indicate attack type: 5'b00000: None 5'b00001: DA = SA 5'b00010: LAND attack (DIP = SIP) 5'b00011: UDP Blat attack (DPORT = SPORT) 5'b00100: TCP Blat attack (DPORT = SPORT) 5'b00101: POD (ping of death) attack 5'b00110: IPv6 fragmented that size is less than the minimum. 5'b00111: ICMP fragmented packet. 5'b01000: ICMPv4 ping that size is over the maximum. 5'b01001: ICMPv6 ping that size is over the maximum. 5'b01010: Smurf (ICMP ping request that DIP is broadcast IP). 5'b01011: TCP header is not complete. 5'b01100: TCP SYN/!ACK packet with sport < 1024. 5'b01101: TCP NULL scan.

		5'b01110: TCP XMAS attack (seq=0, FIN,URG,PSH are set) 5'b01111: TCP SYN/FIN packet (SYN-FIN scan attack). 5'b10000: TCP SYN/RST packet (SYN-RST scan attack). 5'b10001: TCP Fragment Offset = 1 5'b10010: ARP invalid 5'b10011: Gratuitous ARP 5'b10100~5'b11111: Reserved
MAC_CST	1	Indicate this packet meets the MAC constraint condition.
CRC	1	L2/L3 CRC Error Note: The field is a CRC error flag, means that packet can be trapped by other reason but the flag still can be true if the packet is a CRC packet.
SFLOW	6	S-Flow hit port number. 6'b000000 ~ 6'b111101: means the port number that Tx S-Flow hit. 6'b111110: means this packet hit the S-Flow Rx sampling. 6'b111111: Not hit any S-Flow Rx/Tx sampling.
NEW_SA	1	The Source MAC address is a new address and it has been learnt.
STC_L2_PMV	1	The Source MAC causes the port moving of static L2 entry.
OVERSIZE	1	The packet is truncated by NIC (means it's not a completed packet).
REASON	5	0: None (the field is invalid) 1: OAM (OAMPDU,OAM Parser) 2: CFM (CCM Packet,CFM unknown type(opcode),LBM/LBR,LTM/LTR) 3: CFM (ETHDM, It means the field DM_RXIDX is valid) 4: Ingress VLAN filter / Over size (SPG mode) 5: VLAN error(VID=4095 or MBR=0x0) 6: inner/outer CFI=1 7: RMA (User-defined 1) 8: RMA (User-defined 2) 9: RMA (BPDU 01-80-C2-00-00-00) 10: RMA (LACP 01-80-C2-00-00-02) 11: RMA (PTP packet, 0x88F7) 12: RMA (LLDP packet, 0x88CC + 01-80-C2-00-00-0E) 13: RMA (01-80-C2-00-00-xx) 14: IPv6 Hop-by-Hop Extension Header Position Error 15: IPv6 Unknown Extension Header 16: IP4 header error 17: TTL exceed 18: IPv4 options 19: IPv6 header error 20: Hop Limit exceed 21: IPv6 Hop-by-Hop option 22: GW MAC Error/NextHop Age Out 23: IGMP 24: MLD 25: EAPOL 26: ARP Request 27: IPv6 Neighbor Discovery 28: Unknown unicast/multicast trap 29: CPU MAC 30: Invalid SA(MC/BC/All Zero) 31: Normal forwarding (due to L2 table lookup)

When REASON=0, it means the packet is trapped by the other reason. and should refer to *MIR_HIT*, *ACL_HIT*, *ATK_TYPE*, *MAC_CST*, *SFLOW*, *NEW_SA*, and *STC_L2_PMV* fields.

23.2 CPU Tx Tag

The CPU Tx Tag provides control information to instruct switch how to process the packet and it is shown as below:

Table 161: Field Description of CPU Tx Tag

Field Name	Bits	Description
DPM_TYPE	1	Indicate the type of destination port mask: 1'b0: Logical (physical egress port is calculated by trunk module) 1'b1: Physical
ACL_ACT	1	Indicate whether the ingress/egress ACL should take effect.
CNGST_DROP	1	Congestion Droppable. 1'b0: don't drop the packet even the congestion happen. 1'b1: can be dropped when the destination port is in congestion.
BP_FLTR_1	1	Indicate whether to bypass filtering of following modules: (1) Egress OAM mux (2) Egress Port Isolation (3) Egress Mirror Isolation (4) WRED Note: this field takes effect only when AS_DPM=1.
BP_FLTR_2	1	Indicate whether to bypass filtering of following modules: (1) Egress spanning-tree port state (except disabled state) (2) Egress VLAN filtering Note: this field takes effect only when AS_DPM=1.
AS_PRI	1	Priority assignment. 1'b0: the priority is decided by lookup process. 1'b1: assign priority directly.
PRI	3	Assigned priority value (0~7) Note: this field takes effect only when AS_PRI=1.
L2LEARNING	1	L2 learning behavior. 1'b0: don't learn. 1'b1: learn, but still limited by L2 learning constraint.
ALE_AS_TAGSTS	1	Determines the Outer/Inner TAG processing behavior. 1'b0: don't touch VLAN tag. 1'b1: normal processing.
FORWARD_VID_SEL	1	Forwarding VID selection: 1'b0: use inner tag 1'b1: use outer tag
AS_DPM	1	Assign egress ports. 1'b0: follow address table lookup. 1'b1: assign destination ports.
DPM	52	Destination Port Mask (Refer to AS_DPM and DPM_TYPE fields). Note: this field takes effect only when AS_DPM=1.

BP_FLTR_1 and BP_FLTR_2 are useful for some control protocols, such as UDLD(Unidirectional Link Detection), which underlie the logical link layer and would like to ignore the filtering modules.

24 OAM

OAM defines the Operations, Administration and Maintenance sub-layer. It provides mechanisms useful for monitoring link operation such as remote fault indication and remote loopback control. OAM provides network operators the ability to monitor the health of the network and quickly determine the location of failing links or fault conditions.

OAM information is conveyed in Slow Protocol frame called OAM Protocol Data Unit (OAMPDU). OAMPDU contains the appropriate control and status information used to monitor, test and troubleshoot OAM-enabled links.

Dying Gasp is a message sent by the Data Terminating Entity (DTE) when power outage occurs. It is a point-to-point Ethernet OAM communication. The DTE with dying gasp must derive power for a brief period from another source so that the message can be sent without external power.

24.1 Loopback

Loopback mechanism is provided to support a data link layer frame-level loopback mode, which is controlled by remotely. OAM remote loopback can be used for fault localization and link performance testing.

Non-OAMPDU will take the Parser action and the Multiplexer action when enable OAM loopback feature. The MAC address status of loopback packet can be controlled by OAM_CTRL.MAC_SWAP when set the Parser action to loopback. When enable OAM loopback feature, OAMPDU will be trapped to CPU.

Table 162: OAM_PORT_ACT_CTRL Register

Field Name	Bits	Description
PAR_ACT	2	Action to be taken by the Parser for non-OAMPDU ingress traffic. 2'b00: discard; drop all non-OAMPDU ingress traffic 2'b01: forward; pass all non-OAMPDU ingress traffic 2'b10: loopback; pass all non-OAMPDU ingress traffic to Multiplexer for looping it back to the source port (except Pause Frame & CRC error frame). MAC learning must be OFF for looped back traffic on that port. 2'b11: trap
MUX_ACT	1	Action to be taken by the Multiplexer for non-OAMPDU egress traffic. 1'b0: discard; drop all non-OAMPDU egress traffic 1'b1: forward; pass all non-OAMPDU egress traffic

Table 163: OAM_CTRL Register

Field Name	Bits	Description
MAC_SWAP	1	If OAM Loopback is globally enabled and parser is in 'Loopback' state then this bit takes effect. If this bit is enabled then swap the MAC Addresses (Source MAC & Destination MAC) else do not. 1'b0: disable 1'b1: enable
DIS_ACT	1	If EN is set to disable then this bit takes effect. Such as if EN = disable and DIS_ACT = forward then OAMPDU traffic must be forwarded instead of drop. 1'b0: drop 1'b1: forward
EN	1	Globally enable or disable OAM loopback feature. If enable OAM loopback, the receive OAMPDU will be trapped to CPU. 1'b0: disable 1'b1: enable

API REFERENCE

```
rtk_trap_portOamLoopbackParAction_get(uint32 unit, rtk_port_t port, rtk_trap_oam_action_t *pAction);
rtk_trap_portOamLoopbackParAction_set(uint32 unit, rtk_port_t port, rtk_trap_oam_action_t action);

rtk_oam_portLoopbackMuxAction_get(uint32 unit, rtk_port_t port, rtk_action_t *pAction);
rtk_oam_portLoopbackMuxAction_set(uint32 unit, rtk_port_t port, rtk_action_t action);

rtk_oam_loopbackMacSwapEnable_get(uint32 unit, rtk_enable_t *pEnable);
rtk_oam_loopbackMacSwapEnable_set(uint32 unit, rtk_enable_t enable);

rtk_trap_oamPDUAction_get(uint32 unit, rtk_action_t *pAction);
rtk_trap_oamPDUAction_set(uint32 unit, rtk_action_t action);
```

24.2 Dying Gasp

Dying Gasp is a message sent by the DTE when a power outage occurs. It is a point-to-point Ethernet OAM communication. Critical link events are carried within the Flags field of each OAMPDU. Dying gasp is one of critical link event which means an unrecoverable local failure condition has occurred. The DTE with dying gasp must derive power for a brief period from another source so that the message can be sent without external power.

Table 164: OAM_PORT_DYING_GASP_CTRL Register

Field Name	Bits	Description
PORT_OAM_DYING_GASP_EN	1	Enable generating dying gasp message when detected power failure. 1'b0: disable 1'b1: enable

When the time duration that the monitor voltage less than 1.3V is greater than TBP_VAL, dying gasp will be trigged. The monitor voltage must be 1.55V ~ 3.3V in the normal state. When dying gasp is trigged, the device continuous sends out DYING_GASP_PKTCNT packets automatically on enabled dying gasp ports.

The full packet content is per port configured and filled by software in advance through *rtk_oam_dyingGaspPayload_set*. Because the packet content is full controlled by software, it can be standard 802.3ah OAM dying gasp packet or any other message foramt, such as SNMP trap/info, RFC5424 Syslog.

In addition to send dying gasp packets, the device also supports to trigger dying gasp interrupt for software to assert the event.

Table 165: OAM_GLB_DYING_GASP_CTRL Register

Field Name	Bits	Description
DYING_GASP_PKTCNT	3	Select how many dying gasp packets can be sent when power failure happen or trigger dying gasp.
DYING_GASP_TRIG	1	Trigger dying gasp send, auto clear when completed. 1'b0: normal 1'b1: trigger dying gasp manually
TBP_VAL	16	Time of voltage below monitor voltage period. Digital circuit using this register to deglitch. 10ns as counting interval, user can set 10ns-655.35us.

API REFERENCE

*rtk_oam_autoDyingGaspEnable_get(uint32 unit, rtk_port_t port, rtk_enable_t *pEnable);*
rtk_oam_autoDyingGaspEnable_set(uint32 unit, rtk_port_t port, rtk_enable_t enable);
*rtk_oam_dyingGaspPayload_set(uint32 unit, rtk_port_t port, uint8 *pPayload) ;*
*rtk_oam_dyingGaspPktCnt_get(uint32 unit, uint32 *pCnt);*
rtk_oam_dyingGaspPktCnt_set(uint32 unit, uint32 cnt);
*rtk_oam_dyingGaspWaitTime_get(uint32 unit, uint32 *pTime);*
rtk_oam_dyingGaspWaitTime_set(uint32 unit, uint32 time);
rtk_oam_dyingGaspSend_set(uint32 unit, rtk_enable_t enable);

Note: rtk_oam_autoDyingGaspEnable_set must be invoked before rtk_oam_portDyingGaspPayload_set.

25 Automatic Protection Switching

G.8031 and G.8032 are the Ethernet automatic protection switching (APS) mechanisms. The time of protection and recovery switching occupation is within 50ms. It helps achieve highly reliable and stable protection.

G.8031 provides linear protects mechanism in Ethernet layer. It is point-to-point VLAN-based Ethernet sub network connection (SNC). It protects a link between two adjacent nodes using a protection link. It requires a working entity and protection entity. Each bridge selects which link to use to transmit or receive packets.

G.8032 is for Ethernet traffic in a ring topology. It helps achieve never formed loops in the ring. Each Ethernet ring node is connected to adjacent Ethernet ring nodes participating in the same Ethernet ring, using two independent links. The particular link is used to protect the whole ring is called the ring protection link (RPL). Under the normal condition this link is blocked. Under an Ethernet ring failure condition, protection switch blocks traffic on the failure link and the RPL owner node is responsible to unblock RPL, unless the RPL failed.

The device supports link failure detection by sending CFM CCM packet periodically and update keep-alive counter when receive CFM CCM packet by hardware. The mechanism offloads software effort to detect the link failure and therefore gains the time for software to recovery the failed link/ring.

25.1 Link Failure Detection

Configure CFM CCM action to 'LFD' for link failure detect. The device counts down the instance member's keep-alive counter every millisecond and the count down mechanism doesn't start until the CFM CCM packet is received. When receiving CFM CCM packet, the device finds the corresponding instance member by VID and port, then updates its keep-alive counter to the instance's keep-alive timer (CCM_LIFETIME_CTRL.LIFETIME field). Otherwise when the value of keep-alive counter counts down to 0, the device signals the corresponding instance member's interrupt to CPU to notify the link failure event. There is an API ***rtk_oam_linkFaultMon_register*** to register callback function and the registered callback function is invoked when the link failure event is asserted.

Table 166: CFM_CCM_RX_CTRL Register

Field Name	Bits	Description
MD_LV_X	2	Action on receiving CCM frames with MD Level X. (X = 0~7) 2'b00: drop 2'b01: trap to CPU 2'b10: forward 2'b11: LFD

The device supports 8 ERP instances. The configurations for Rx instance are member port, VID and keep-alive timer (CCM_LIFETIME_CTRL.LIFETIME field). The keep-alive timer is used to update the value of keep-alive counter by hardware when receive CCM packet which corresponding the member of Rx instance (port and VID). The CCM_RX_INST_CTRL and CCM_LIFETIME_CTRL registers are per instance configurations.

Table 167: CCM_RX_INST_CTRL Register

Field Name	Bits	Description
VID	12	VID associated with the corresponding instance. VID = 0 signifies untagged CCM frames are permitted and do not check
P1	6	Port number which would be participated in ERP and would receive CCM frames. Set to 0x3F to disable participating in ERP.
P0	6	Port number which would be participated in ERP and would receive CCM frames. Set to 0x3F to disable participating in ERP.

Table 168: CCM_LIFETIME_CTRL Register

Field Name	Bits	Description
LIFETIME	10	Lifetime value which must be used to update the ERP instance member's keep-alive counters upon receiving a CCM frame for the corresponding instances.

The device sends a CCM packet per configured transmit interval of instance. User can configure the transmitted packet content including tag status, tag information, MEP ID, CCM interval, MD level and MAID. The tag status is whether transmitted packet should be tagged or not. The tag information includes TPID, PCP, CFI and VID. The other configurable fields of Tx instance are transmit status, transmit interval and member port. The CCM_TAG_CTRL and CCM_TX_CTRL registers are global configurations while CCM_TX_INST_CTRL and CCM_TX_INST_P_CFG registers are per instance configurations.

Table 169: CCM_TAG_CTRL Register

Field Name	Bits	Description
PCP	3	Priority Code Point value of the tag to be used in the generated CCM frames.
CFI	1	Canonical Format Identifier of the tag to be used in the generated CCM frames.
TPID	16	Tag protocol Identifier for the generated CCM frames.

Table 170: CCM_TX_CTRL Register

Field Name	Bits	Description
MEPID	13	MEPID to be inserted into the generated CCM frame
LIFETIME_CODE	3	Lifetime code to be inserted into the generated CCM frame
MDL	3	MD Level to be inserted into the generated CCM frames

Table 171: CCM_TX_INST_CTRL Register

Field Name	Bits	Description
VID_ADD	1	This bit signifies whether VLAN tag would be inserted into the CCM frames generated by PORT0 & PORT1. If enable then VLAN tag would be inserted into the CCM frames else not. 0b0: disable ob1: enable
VID	12	VID to be added into the CCM frames if VID_ADD bit is enabled.
MAID	8	LSB 1 byte to be inserted into the generated CCM frame as MAID
TX	1	This bit enables/disables CCM frame generation & transmission corresponding to the instance. 0b0: disable, do not generate CCM packets 0b1: enable, enable CCM packet generation
INTLV	10	CCM frame transmission interval for the corresponding instances. (msec)

Table 172: CCM_TX_INST_P_CFG Register

Field Name	Bits	Description
P1	6	Port number which would be participated in ERP and would generate & send CCM frames. Set to 0x3F to disable participating in ERP.
P0	6	Port number which would be participated in ERP and would generate & send CCM frames. Set to 0x3F to disable participating in ERP.

API REFERENCE

```
rtk_trap_cfmCcmAct_get(uint32 unit, uint32 level, rtk_trap_oam_action_t *action);
rtk_trap_cfmCcmAct_set(uint32 unit, uint32 level, rtk_trap_oam_action_t action);

rtk_oam_cfmCcmRxInstVid_get(uint32 unit, uint32 instance, rtk_vlan_t *vid);
rtk_oam_cfmCcmRxInstVid_set(uint32 unit, uint32 instance, rtk_vlan_t vid);

rtk_oam_cfmCcmRxInstPort_get(uint32 unit, uint32 instance, uint32 index, rtk_port_t *port) ;
rtk_oam_cfmCcmRxInstPort_set(uint32 unit, uint32 instance, uint32 index, rtk_port_t port) ;

rtk_oam_cfmCcmResetLifetime_get(uint32 unit, uint32 instance, uint32 *lifetime) ;
rtk_oam_cfmCcmResetLifetime_set(uint32 unit, uint32 instance, uint32 lifetime) ;

rtk_oam_cfmCcmPcp_get(uint32 unit, uint32 *pcp);
rtk_oam_cfmCcmPcp_set(uint32 unit, uint32 pcp);

rtk_oam_cfmCcmCfi_get(uint32 unit, uint32 *cfi);
rtk_oam_cfmCcmCfi_set(uint32 unit, uint32 cfi);

rtk _oam_cfmCcmTpid_get(uint32 unit, uint32 *tpid);
rtk _oam_cfmCcmTpid_set(uint32 unit, uint32 tpid);

rtk _oam_cfmCcmMepid_get(uint32 unit, uint32 *mepid);
rtk _oam_cfmCcmMepid_set(uint32 unit, uint32 mepid);

rtk_oam_cfmCcmIntervalField_get(uint32 unit, uint32 *interval);
rtk_oam_cfmCcmIntervalField_set(uint32 unit, uint32 interval);

rtk _oam_cfmCcmMdl_get(uint32 unit, uint32 *mdl);
rtk _oam_cfmCcmMdl_set(uint32 unit, uint32 mdl);

rtk_oam_cfmCcmInstTagStatus_get(uint32 unit, uint32 instance, rtk_enable_t *enable);
rtk_oam_cfmCcmInstTagStatus_set(uint32 unit, uint32 instance, rtk_enable_t enable);

rtk_oam_cfmCcmInstVid_get(uint32 unit, uint32 instance, rtk_vlan_t *vid);
rtk_oam_cfmCcmInstVid_set(uint32 unit, uint32 instance, rtk_vlan_t vid);

rtk_oam_cfmCcmInstMaid_get(uint32 unit, uint32 instance, uint32 *maid);
rtk_oam_cfmCcmInstMaid_set(uint32 unit, uint32 instance, uint32 maid);

rtk_oam_cfmCcmInstTxStatus_get(uint32 unit, uint32 instance, rtk_enable_t *enable);
rtk_oam_cfmCcmInstTxStatus_set(uint32 unit, uint32 instance, rtk_enable_t enable);

rtk_oam_cfmCcmInstInterval_get(uint32 unit, uint32 instance, uint32 *interval);
rtk_oam_cfmCcmInstInterval_set(uint32 unit, uint32 instance, uint32 interval);

rtk _oam_cfmCcmTxInstPort_get(uint32 unit, uint32 instance, uint32 index, rtk_port_t *port) ;
rtk _oam_cfmCcmTxInstPort_set(uint32 unit, uint32 instance, uint32 index, rtk_port_t port) ;

rtk_oam_linkFaultMonEnable_set(rtk_enable_t enable);
rtk_oam_linkFaultMon_register(rtk_oam_linkFaultMon_callback_t callback);
rtk_oam_linkFaultMon_unregister(void);
```

PROGRAMMING EXAMPLE

Link failure detects in port 3 and 4 of VLAN 2.

```
int32 linkFailureHandler(rtk_oam_linkFaultEvent_t * event);

level = 3;
instance_id = 1;
vid = 2;

/* Configure CCM packet for link failure detect */
rtk_trap_cfmCcmAct_set(unit, level, TRAP_OAM_ACTION_LINK_FAULT_DETECT);

/* Rx */

/* Configure Rx link failure detect in VLAN 2 */
rtk_oam_cfmCcmRxInstVid_set(unit, instance_id, vid);
/* Configure Rx link failure detect in port 3 and 4 */
rtk_oam_cfmCcmRxInstPort_set(unit, instance_id, 0, 3) ;
rtk_oam_cfmCcmRxInstPort_set(unit, instance_id, 1, 4) ;
/* Confgiure keep-alive timer to 10 ms */
rtk_oam_cfmCcmResetLifetime_set(unit, instance_id, 10);
/* Register callback function to be invoked when link failure event is asserted */
rtk_oam_linkFaultMon_register(linkFailureHandler);
/* Enable CallBack */
rtk_oam_linkFaultMonEnable_set(ENABLED);

/* Tx */

/* Configure Tx link failure detect in port 3 and 4 */
rtk_oam_cfmCcmTxInstPort_set(unit, instance_id, 0, 3);
rtk_oam_cfmCcmTxInstPort_set(unit, instance_id, 1, 4);

/* Tag content */
/* Transmit CCM packet with tag */
rtk_oam_cfmCcmInstTagStatus_set(unit, instance_id, ENABLED);
/* Configure tag information */
/* Configure transmitted CCM packet TPID */
rtk_oam_cfmCcmTpid_set(unit, 0x8898);
/* Configure transmitted CCM packet PCP */
rtk_oam_cfmCcmPcp_set(unit, 5);
/* Configure transmitted CCM packet CFI */
rtk_oam_cfmCcmCfi_set(unit, 0);
/* Configure transmitted CCM packet VID */
rtk_oam_cfmCcmInstVid_set(unit, instance_id, vid);

/* CFM CCM content */
/* Configure transmitted CCM packet MD level */
rtk_oam_cfmCcmMdl_set(unit, level);
/* Configure transmitted CCM packet CCM interval */
rtk_oam_cfmCcmIntervalField_set(unit, 2);
/* Configure transmitted CCM packet MEP ID */
rtk_oam_cfmCcmMepid_set(unit, 6);
/* Configure transmitted CCM packet MAID */
rtk_oam_cfmCcmInstMaid_set(unit, instance_id, 5);

/* transmit CCM packet every 3 ms*/
rtk_oam_cfmCcmInstInterval_set(unit, instance_id, 3);
/* Start to transmit CCM packet */
rtk_oam_cfmCcmInstTxStatus_set(unit, instance_id, ENABLED);
```

26 Cable Diagnostic

Cable diagnostic is supported by “RTCT” (RealTek Cable Tester) feature of Realtek PHYs. RTCT can detect open, short or normal in each differential pairs with cable length information. It transmits a signal of known amplitude to one pair of an attached cable. The transmitted signal will continue down the cable until it reflects off a cable imperfection. The magnitude of the reflection and the time it takes for the reflection to come back is then used to locate the type of problem and the distance to the problem.

For a normal link up port, its port state is linked down when RTCT starts and the link state is recovered after RTCT stops. If links down is not desirable for a normal link up port, there is another feature “Green Ethernet” can be used to retrieve the cable length.

26.1 RTCT

RTCT flow chart is shown as below:

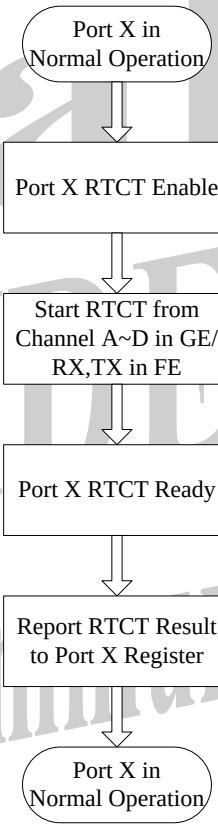


Figure 52: RTCT Flow Chart

Four states are supplied by RTCT:

- Short

A short is detected on the cable.

- Open

An opening is detected on the cable. One scenario is the cable doesn’t plug to the link partner.

- Impedance Mismatch

The impedance is mismatched. The normal impedance should be 100Ω, impedance mismatch is detected if the

impedance measured is not in the range 70Ω~130Ω.

- Line Driver

The high impedance is detected. One scenario is the cable plug to a power down link partner.

The cable length reported is the length where Short/Open/Impedance Mismatch/Line Driver is detected and **the cable length reported by RTCT could have a plus or minus 3 meters inaccuracy.**

API REFERENCE

*rtk_diag_rtctEnable_set(uint32 unit, rtk_portmask_t *pPortmask);*
*rtk_diag_portRtctResult_get(uint32 unit, rtk_port_t port, rtk_rtctResult_t *pRtctResult);*

26.2 Green Ethernet

Green Ethernet is another feature that can be used to retrieve the cable length when the port is linked on. **The cable length reported by Green Ethernet could have a plus or minus 15 meters inaccuracy.**

API REFERENCE

*rtk_diag_rtctEnable_set(uint32 unit, rtk_portmask_t *pPortmask);*
*rtk_diag_portRtctResult_get(uint32 unit, rtk_port_t port, rtk_rtctResult_t *pRtctResult);*

**The API inside automatically switch to Green Ethernet if the diagnostic port to be checked is link up.*