MPLS-TP OpenFlow Protocol Extensions for SPTN

Version 1.0
June 16, 2017

ONF TS-029

ONF Document Type: Draft Technical Specification
ONF Document Name: SPTN OpenFlow Protocol Extensions

Disclaimer

Table of Contents

List of Figures

List of Tables

1 Overview

1.1 Scope and Objectives

This document specifies the abstract objects and entities that constitute the semantics of the SPTN Southbound Interface (SBI).

This specification is based on ONF standards, specifically: OpenFlow 1.3 [3] with the ONF Extensions Package [7], OF-CONFIG 1.2 [2], and OpenFlow Table Type Patterns [5]. In addition it incorporates the egress table concept, introduced in OpenFlow 1.5 [5], as an OpenFlow 1.3 extension to support Ethernet OAM.

The OpenFlow specification describes the basic components and functions of an OpenFlow Logical Switch (OFLS) along with the OpenFlow switch protocol used to remotely manage it from an OpenFlow controller. The OpenFlow protocol can thus be viewed as providing the syntax notation for programming a packet processing pipeline.

This SPTN OpenFlow specification defines the abstract objects and entities for a specific type of OFLS that provides the features required of a Packet Transport Network (PTN) switch node as described in [9]. In order to meet these requirements, this specification defines a number of Experimenter extensions to the basic OpenFlow feature set. These extensions include new match fields, actions, action tables, and counter objects, where action tables implement parameterized actions. In particular, action tables can be used in group bucket action sets as well as flow rules.

The OpenFlow protocol assumes the OFLS has been configured with various artifacts. The OpenFlow Configuration Protocol (OF-CONFIG) has been defined to enable remote configuration of an OpenFlow Capable switch (OFCS) from an OpenFlow Control Point (OFCP). An OFCS can contain one or more OFLS instance. This SPTN specification defines a number of new artifacts, primarily for OAM, QoS, and notification and alarms. These follow the ONF information modeling framework and, as such, are compatible with OF-CONFIG 1.2 [2]. This is also described in Section 2.

PTN equipment consists of a data plane, control plane, and management plane. The SPTN OFLS described in this document corresponds to the data plane. An application on the OpenFlow Controller provides the control plane, and communicates with the OFLS using the OpenFlow protocol. The OFCP provides the management plane by communicating with the OFCS using OF-CONFIG. An SPTN OFCS contains a single OFLS instance.

1.2 Common Terms, Abbreviations and Definitions

The following terms, definitions, and abbreviations are used in this standard.

Term	Explanation
AC	Attachment circuit
ACH	Associated Channel Header
ACL	Access Control List

APS	Automatic Protection Switching
BOS	Bottom of Stack
CAR	Committed Access Rate
CBS	Committed Burst Size
CE	Customer Edge
CIR	Committed Information Rate
CW	Control Word
DiffServ	Differentiated Services
DSCP	Differentiated Services Code Point
EBS	Excess Burst Size
ECMP	Equal Cost Multi Path, used for multi-path routing
E-LSP	EXP-Inferred-PSC LSP
ENNI	External Network-Network Interface. Logical demarcation point between two operator networks.
EXP	MPLS field that denotes LSP PHB (priority). See MPLS_TC.
Flow	Sequence of packets with the same selection of header field values. Flows are unidirectional.
Flow Table	OpenFlow flow match-action table as defined in the OpenFlow specification.
Flow Entry	Entry in an OpenFlow flow table with its match fields and instructions that conforms to the OpenFlow specification.
G-ACh	Generic Associated Channel
GAL	G-ACh Label
Group Table	OpenFlow group table as defined in the OpenFlow specification.
Group Table Entry	Entry in an OpenFlow group table, with its Group Identifier, Group Type, Counters and Action Buckets fields.
IN_PORT	Ingress Port
LSP	Label Switched Path
LSR	Label Switched Router
MP2MP	Multipoint-to-Multipoint
MPLS	Multi-Protocol Label Switching
MPLS_TC	MPLS field that denotes LSP PHB (priority).
MPLS-TP	Multi-Protocol Label Switching – Transport Profile

NNI	Network-to-Network Interface. In this specification, a logical demarcation point in an operator network.
OAM	Operation, Administration, and Maintenance
OF-CONFIG	OpenFlow Configuration Protocol
OFCP	OpenFlow Configuration Protocol
OFCS	OpenFlow Capable Switch
OFLS	OpenFlow Logical Switch
ONF	Open Networking Foundation
OpenFlow Protocol	A communications protocol between control and data plane of supported network devices, as defined by ONF.
P2MP	Point-to-Multipoint
P2P	Peer-to-Peer
PE	Provider Edge device.
PHB	Per-hop behavior
PIR	Peak Information Rate
PSC	PHB Scheduling Class. A set of PHB(s) which share the same ordering constraint.
PTN	Packet Transport Network
PW	Pseudo-wire
QoS	Quality of Service
SD	Signal Degrade indication (APS)
SD-Tag	Service delimiting VLAN tag
SF	Signal Fail indication (APS)
SPTN	Super PTN
SBI	Southbound Interface
SDN	Software Defined Networking.
SP	Strict Priority
trTCM	Two-rate three-color marking
TTP	Table Type Pattern
UNI	User Network Interface. A demarcation point between a subscriber and the service provider PE for Ethernet service delivery.
VPN	Virtual Private Network
VPWS	Virtual Private Wire Service

VPLS	Virtual Private LAN Service
WFQ	Weighted Fair Queuing
WRED	Weighted Random Early Discard

1.3 Normative Language

The key words "MUST","MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

2 SBI Requirements

2.1 OpenFlow

The SPTN OFLS is specified using the ONF TTP model [5]. The OpenFlow 1.3.4 [3] specification is very general and includes many optional features. It also includes an extensions facility. The TTP approach was used to describe a specific OFLS in terms of specific flow tables and group entries, what match fields and actions each supports, how they are interrelated, and packet flows. In addition, it describes extensions and how and where they are incorporated. This document also provides the Experimenter protocol encodings required for programming these extensions in Section 9.

2.1.1 Design Principles

The SPTN OFLS makes use of multiple tables to:

- Factor the forwarding functions into manageable components for efficient programming; and
- Map to existing as well as future hardware for multiple types of network elements, for example, multi-card chassis implementations.

The SPTN TTP model is logically organized into stages as shown in Figure 1.

Figure 1 TTP Organizational Principles

The first stage consists of the packet processing match-action tables that decide how to further process and forward the packet. These tables are also where encapsulation header and tag removal is performed using OpenFlow Apply-Actions instructions and action lists.

The next stage has the forwarding group table entries that execute the forwarding decision by taking advantage of OpenFlow multi-bucket group types to perform multicast replication (using the All type), multipath load balancing (using the Select type), and protection (using the Fast Failover type).

Once the output port is decided for the packet or for a replica, packet editing groups (using the indirect type) are used to perform such functions as: adding encapsulation headers, pushing tags, updating header fields, etc.[1] This stage also includes egress tables, which are applied between the forwarding groups and the output port.[2]

Note that all tables have explicit built-in table miss rules. This specification does not use global or per-table configuration settings for the case of a lookup miss.

The normative SPTN TTP is described in detail in the JSON file referenced in Section 10 of this specification.

2.2 Configuration

The SPTN configuration is described using an information model expressed as a UML diagrams. The UML diagrams are used to derive IETF YANG models [23] for use with OF-CONFIG. Should the diagrams and YANG models differ, the YANG files MUST be considered normative.

The YANG files are referenced in Sections 8 and 11 of this specification.

2.3 Document Structure

This document is structured according to SPTN functional requirements as follows:

- Section 3 describes the forwarding Service Model.
- Section 4 describes the QoS module.
- Section 5 describes OAM.
- Section 6 describes Protection (APS).
- Section 7 describes Statistics.
- Section 8 references notification and alarm extensions as augments to OF-CONFIG. The YANG text file that accompanies this specification SHALL be considered the normative notification and alarm configuration specification.
- Section 9 provides detailed Experimenter encodings for OpenFlow extensions described in this document. This entire section MUST be interpreted as normative.
- Section 10 references the TTP model and provides an informative diagram showing all tables, groups, and objects of the TTP, with their interconnections. The JSON text file that accompanies this specification MUST be considered the normative TTP specification.
- Section 11 references OAM and QoS extensions as augments to OF-CONFIG. The YANG text files that accompany this specification MUST be considered the normative OAM and QoS configuration specifications.

[1] This specification also makes use of egress match action tables in the context of the output port.
[2] Egress tables are an OpenFlow extension discussed in Section 5.3.1

Sections 3 through 7 describe the tables and objects that implement specific aspects of this specification. These sections provide informative descriptive content.

3 Service Model

This specification supports Peer-to-Peer (P2P) (E-Line) L2 Virtual Private Network (VPN) services. Subsequent versions will support additional L2 VPN services, including Multipoint-to-Multipoint (MP2MP) (E-LAN) and Point-to-Multipoint (P2MP) (E-Tree), and also L3 VPN services.

3.1 Requirements

The L2 VPN service follows the framework defined in RFC4664 for Virtual Private Wire Service (VPWS). The E-Line service supports service channel to pseudo-wire (PW) bundling based on port and on port and VLAN. For this purpose, the OpenFlow switch objects map a customer attachment circuit (AC) to a single PW by matching on ingress port (IN_PORT) or on (IN_PORT, VLAN-ID).

3.2 Packet Flows

3.2.1 VPWS

This section describes the abstract object pipeline for VPWS in terms of packet flows between the customer User-network Interface (UNI) and the provider-facing Network-to-Network Interface (NNI). These abstract objects consist of flow tables, group table entries, and ports.

3.2.1.1 *Attachment Circuit (UNI->NNI)*

The flow tables for processing client frames are shown in Figure 2. Client frames ingress from UNI physical ports in the direction of NNI physical ports.

In this and subsequent diagrams, Flow Tables are shown outlined with square corners and colored in rose. Group Table entries are shown with rounded corners and grey coloring.

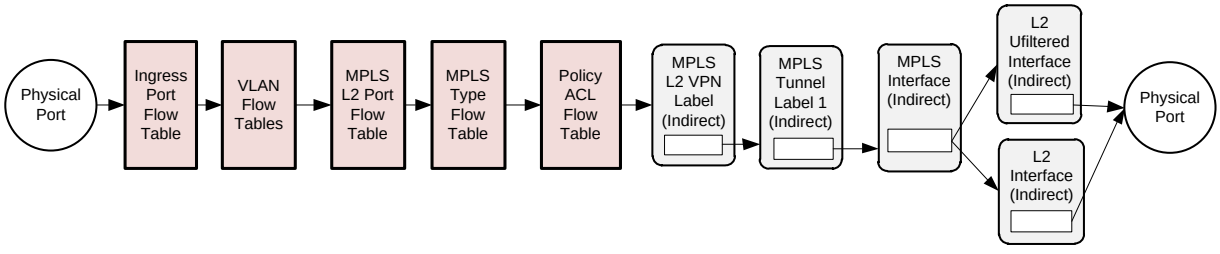

Figure 2 VPWS UNI->NNI Packet Processing

The flow tables used for UNI->NNI packet flows factor the pipeline packet processing functions as follows:

- **Ingress Port Flow Table**. OpenFlow requires the first table in the pipeline to be numbered 0. This table is used to dispatch different packet types to specialized pipelines. Default rules are used to simplify programming.

- **VLAN Flow Table**. This table is used to classify packets into different logical streams by matching primarily on the ingress port and/or the contents of VLAN tags, and assigning metadata to control subsequent processing. VLAN Flow Table flow entries that match customer frames have a Go-to Table instruction specifying the Multi-Protocol Label Switching (MPLS) L2 Port Flow Table. These flow entries can set properties about the flow. One such property is the MPLS_TYPE pipeline field, which is used to distinguish VPWS from VPLS flows.
- **MPLS L2 Port Flow Table**. This table represents a logical ingress interface using metadata assigned in the classification step. VPWS forwarding is completely determined by this table. This table is also used for purposes such as OAM and QoS that will be described in subsequent sections. MPLS L2 Port Flow Table flow entries can have a Go-to Table instruction specifying the MPLS Type Flow Table and a Write-Actions instruction that adds a group entry to the action set.
- **MPLS Type Flow Table**. This table matches on the MPLS_TYPE pipeline field to direct MPLS packets to different forwarding tables by matching on the MPLS_TYPE pipeline field. This is a fixed logic table where all rules are built-in; the Controller cannot add, delete, or modify rules in this table. The default on a miss (unrecognized MPLS_TYPE) is to drop the packet, that is, to execute a Clear Actions instruction which nullifies any group or output action. Flow entries in this table always have a Go-to Table instruction specifying the Policy ACL Flow Table.
- **Policy ACL Flow Table**. This table provides the flexibility to perform the kinds of security filtering actions that are typically done using ACLs. If the Policy ACL Flow table has no Go-to Table instruction, the packet will be processed for forwarding by the group entry or entries referenced in the action set. If there is no group or output action the packet is dropped.

This specification uses a concept of typed group entries in the group table. The group entry type determines properties of the group, including its OpenFlow type, constraints on the contents of its action buckets, and how the packet is processed next. It effectively partitions the OpenFlow monolithic group table into logical sub-tables.

The following group table entry types are used in UNI->NNI packet flows:

- **MPLS L2 VPN Label**. This indirect group entry type contains a single bucket whose action set includes: push an outer Ethernet header onto the frame; push an MPLS shim header; set the label value and bottom of stack in the header to the PW label; and push a Control Word (CW). It also contains actions to set the TTL and MPLS_TC fields as needed. By the OpenFlow specification, the push inserts an MPLS shim header as the outermost MPLS label, in this case between the new and old L2 headers.

  The L2 header and CW push actions are extensions to support PTN. The CW is initialized to zero (sequence numbering is not supported). This group chains to an MPLS Tunnel Label 1 group entry type.
- **MPLS Tunnel Label 1**. This indirect group entry type is used to push another outermost MPLS shim header and set the value of the LSP label and associated fields. This group normally chains to an MPLS Interface group entry.

- **MPLS Interface**. This indirect group sets the MAC-DST, MAC-SRC, and VLAN ID for forwarding the packet to the next hop node, typically a Label Switched Router (LSR). This group can chain to either an L2 Interface group or an L2 Unfiltered Interface group entry type.
- **L2 Interface/L2 Unfiltered Interface**. These groups are used to specify an output port and configure its VLAN properties. An L2 Interface group entry type is used for VLAN filtering and tagging, whereas an L2 Unfiltered Interface indicates a port that does not do VLAN filtering. Typically an L2 Interface is used to filter customer VLANs for UNI ports and an L2 Unfiltered interface is used for NNI ports, although either type could be used in either case. This group has an output action to a physical port.

Note that multiple group entries are used for MPLS labels in order to accommodate the fact that the OpenFlow specification only permits a single action of any particular type in an action set. Hence multiple group entries are used to push multiple labels.

3.2.1.2 *Network Interface (NNI->UNI)*

The flow tables used for processing frames received from the network are shown in Figure 3. Network frames ingress from NNI physical ports in the direction of UNI physical ports.

Two MPLS Flow Tables are typically used for processing LSP and PW labels.

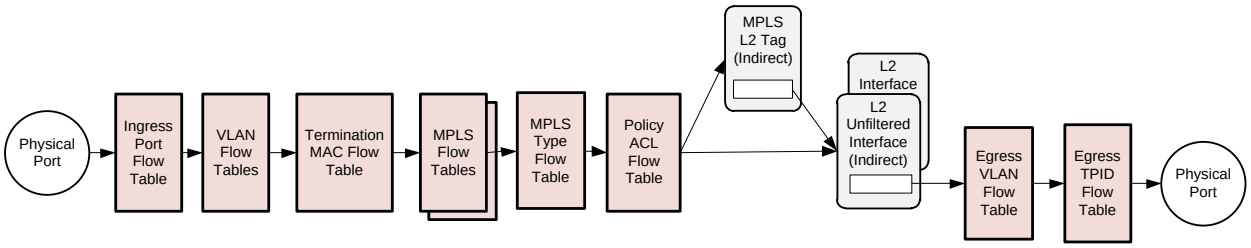

Figure 3 VPWS NNI->UNI Packet Processing

The flow tables used for NNI->UNI packet flows factor the pipeline packet processing functions as followings:

- **Ingress Port Flow Table**. Same as previously described.
- **VLAN Flow Table**. This table provides VLAN filtering on network frames, if required. Packets always pass through the VLAN Flow Table, although it can be programmed to allow any VLAN ID. The VLAN Flow Table should be programmed so that flow entries that match NNI ports have a Go-to Table instruction specifying the Termination MAC Flow Table.
- **Termination MAC Flow Table**. This table is used to recognize MPLS frames by matching on ingress port, Ethertype, destination MAC address, and optionally VLAN ID, and steer them for MPLS processing. Unmatched frames are dropped.
- **MPLS Flow Tables**. Two MPLS Flow Tables are used to process incoming labels. Multiple tables are used since OpenFlow is only able to match the outermost shim header.

  Each table can match on ingress port (optional), Ethertype (to satisfy OpenFlow prerequisites), MPLS label, and MPLS bottom of stack (BOS). Flow entries can apply actions based on the label, including: pop, pop and forward based on this label, or pop with a

Goto-Table instruction specifying the second MPLS Flow Table.

Flow entries that pop the PW label include additional actions to pop the outer L2 header and to pop the CW. VPWS forwarding includes a Group action in the action set[3], in which case MPLS_TYPE must be set to VPWS.

- **MPLS Type Flow Table**. Same as previously described.
- **Policy ACL Flow Table**. Same as previously described.
- **Egress VLAN Flow Table**. This table can be used to modify the packet VLAN tags for packets output by an L2 Unfiltered Interface group entry. The outgoing frame can be untagged, single tagged, or double tagged.
- **Egress TPID Flow Table**. This table can optionally change the TPID on the outermost VLAN tag to be an S-Tag. It might be used, for example, in a UNI-UNI interoperability use case. By default the outermost tag is a C-Tag.

Note that by design, the contents of the MPLS Flow Tables must be kept in agreement. OpenFlow 1.4 introduces a feature (synchronized tables) that supports this. Synchronized tables can be used with OpenFlow 1.3.4 protocol using the Table Synchronization extension (EXT-232).

The following group table entry type is used in NNI->UNI packet flows:

- **MPLS L2 Tag**. This group can optionally push an outer service delimiting (SD) VLAN tag [27].
- **L2 Interface/L2 Unfiltered Interface**. These groups are used to specify an output port and configure its VLAN properties. An L2 Interface group entry type is typically used for VLAN filtering and specifying tagging on the physical port. An L2 Unfiltered Interface should be used if the EgressVLAN Flow Table applies VLAN translation .

3.2.2 LSR

The flow tables used for processing network to network frames are shown in Figure 4. Network frames ingress from NNI physical ports and are forwarded to NNI physical ports.

In addition to the group entry that performs swap or push up to two other label group entries are optionally available. These are shown outlined using dotted lines.

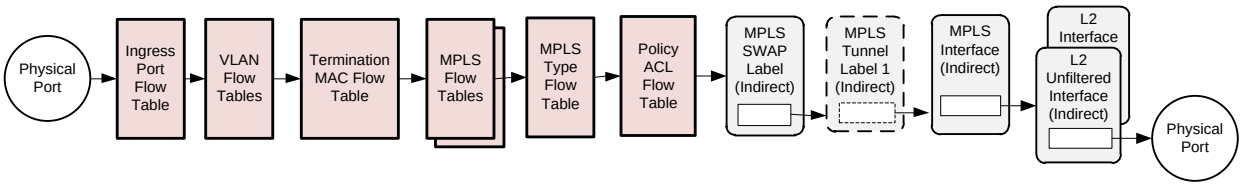

Figure 4 LSR Packet Processing

The flow tables used for NNI->NNI packet flows factor the pipeline packet processing functions as follows:

[3] This specification does not check the sequence number in the CW.

- **Ingress Port Flow Table**. Same as previously described.
- **VLAN Flow Table**. Same as for the NNI->UNI packet flow.
- **Termination MAC Flow Table**. Same as for the NNI->UNI packet flow.
- **MPLS Flow Tables**. These are used to process incoming label stacks. Each can match on ingress port (optional), Ethertype (to satisfy OpenFlow prerequisites), MPLS label, and MPLS BOS. They can apply actions to pop MPLS shim headers and perform actions based on the label. Actions include pop, pop and forward based on this label, or pop and forward based on an inferior label. Forwarding is accomplished by including a Group action in the action set. Depending on the particular flow, the Group can be an MPLS SWAP label or MPLS L2 VPN label type.
- **MPLS Type Flow Table**. Same as previously described.
- **Policy ACL Flow Table**. Same as previously described.

The following group table entry types are used for NNI->NNI packet flows:

- **MPLS L2 SWAP Label.** This indirect group entry type is used to set the label value in the existing outermost MPLS shim header. It also contains actions to update the TTL and MPLS_TC fields as needed. This group can chain to either an MPLS Interface or an MPLS Tunnel Label 1 group entry type.
- **MPLS L2 VPN Label**. Same as previously described.
- **MPLS Tunnel Label 1**. This indirect group entry type is used to push another outermost MPLS shim header and set the value of the LSP label and associated fields. The dotted lines are to show that use of this group is optional. This group can chain to an MPLS Interface group entry or to an optional MPLS Tunnel Label 2 group entry.
- **MPLS Interface**. Same as previously described.
- **L2 Interface/L2 Unfiltered Interface**. Same as previously described.

3.3 OpenFlow Extensions

OpenFlow 1.3.4 requires extensions in order to support MPLS-TP. These are described in this section. The Experimenter protocol encodings for these extensions are provided in Section 8.

3.3.1 Match Fields

VPWS does not require additional protocol header match fields, although such fields may be introduced in the future, e.g., to support CW sequence numbers.

3.3.2 Pipeline Match Fields

OpenFlow 1.3.4 [3] added a clarification for Pipeline Match fields. Pipeline Match fields are metadata fields, such as TUNNEL_ID, that can be both matched in flow entries and set using Set-Field actions.[4]

[4] This use of TUNNEL_ID is non-standard but acceptable.

The following new pipeline match fields have been identified.

- **MPLS_L2_PORT.** Identifies a logical interface to facilitate VPWS forwarding.
- **MPLS_TYPE**. The only valid value is VPWS(1).
- **ALLOW_VLAN_TRANSLATION**. Matched in Egress VLAN Table entries that do VLAN translation.

These fields are used in VLAN, MPLS_L2_PORT, and Egress VLAN Flow Table flow entries.

3.3.3 Actions

The following new actions are needed.

- **PUSH_CW/POP_CW_OR_ACH**. Push and pop a control word (CW) or associated channel header (ACH). This field is located immediately after the bottom-of-stack MPLS label. PUSH_CW inserts a 32-bit word immediately after the outermost MPLS shim header. PUSH_CW should only be used when there is a single MPLS header with bottom-of-stack set. POP_CW_OR_ACH removes a 32-bit word following the outermost MPLS header which must have bottom-of-stack set. If bottom-of-stack is not set the actions are undefined and likely a no-op. For this specification the CW is always initialized to zero. Sequence numbering is not supported.
- **PUSH_L2_HDR/POP_L2_HDR**. Add or remove an outer Ethernet encapsulation header. The header is 14-bytes and has MAC_SRC, MAC_DST, and ETH_TYPE fields. When pushed an Ethertype argument is required and MAC_SRC and MAC_DST are initialized to zero. A VLAN tag can be pushed using existing actions.

In general, pop actions are performed in flow table action lists using the Apply-Actions instruction. Push actions are performed in group-bucket action sets.

3.4 Configuration Extensions

No configuration extensions are required for the service model.

4 QoS

This section describes the SPTN logical switch objects supporting QoS. Note that OpenFlow 1.3.4 does not fully support the metering and marking capabilities required. This section discusses QoS concepts and requirements followed by the packet flow diagrams.

4.1 QoS Concepts

QoS packet processing follows the DiffServ Pipe model described in RFC3270, which comprises the following:

- Classification - Assigning a priority and drop precedence. Simple classification is based on matching those packet header fields specifically designated for QoS marking. Complex classification implements more complicated rules involving multiple header fields.

- Metering – Policing, which can change the drop precedence based on flow properties such as packet and byte rate.
- Marking - Setting QoS fields in the packet headers based on priority and traffic class.
- Shaping – Queuing or dropping the packet, based on traffic class and priority, respectively. Queues are serviced-based on the scheduling discipline. Typically queues implement an admission scheme based on drop precedence, such that they stop accepting packets with higher drop precedence when the number of entries in the queue exceeds some threshold.

To support classification, OpenFlow is extended with new pipeline-match fields for Traffic Class (for priority) and Color (for drop precedence). Eight Traffic-Class values are used to support the following PHB groups: BE (0), AF1 (1), AF2 (2), AF3 (3), AF4 (4), EF (5), CS6 (6), and CS7 (7). Color can take one of three values: Green (0, in-profile), Red (2, out of profile and drop), and Yellow (1, out of profile but lower precedence than green packets).

To support metering the OpenFlow Meter table is extended with additional features. The requirements are to support RFC 2698 two-rate three-color marking (trTCM) in both color-blind and color-aware modes. For this, a new type of Meter Table entry is defined that contains two meter bands. Each meter band indicates actions to apply immediately to the packet. In OpenFlow 1.3.4, the meter instruction must be evaluated before other instructions.[5]

A new Color Meter Band type repurposes OpenFlow rate and burst parameters to configure token bucket marker algorithms. For example, for trTCM, the Yellow band rate is interpreted as Committed Information Rate (CIR), the rate at which tokens are added, and burst is interpreted as Committed Burst Size (CBS), or the bucket size. For the Red band, rate is Peak Information Rate (PIR) and Excess Burst Size (EBS). The Color Meter Band type has a single action to change the packet Color pipeline match field, depending on which band the packet falls in. Different bucket types are defined to support color-aware or color-blind operation, as well as selection of trTCM (RFC 2698), modified trTCM (RFC 4115) or srTCM (RFC 2697). These are used instead of the standard OpenFlow meter bands (Drop, DSCP Remark), which are not used here.

To support marking, the L2 Policer-Actions Flow Table is placed after the L2 Policer Flow Table, since the results of applying a meter cannot be realized in the same table that applies the meter. This can be used for DSCP and VLAN priority marking. To mark MPLS shim headers that are subsequently pushed onto the frame, MPLS Label Re-Mark Tables are introduced and invoked as actions from the MPLS Label group entry action sets.

Queuing occurs at the egress port after all packet processing. Since in OpenFlow queuing is considered switch dependent, this specification defines a specific queue behavior. Eight queues per port are provided, with each queue corresponding to a PHB priority class. Furthermore, the output queue is determined by the value of the packet Traffic Class at egress and not by a Set-Queue action. The OpenFlow Set-Queue action is not used.

[5] In the future OpenFlow may deprecate the meter instruction in favor of a meter action. This would not materially change the operation of meters but will change how meters were programmed from the controller.

These queues can be configured in terms of minimum and maximum rates using OF-CONFIG. In addition, queues are enhanced with congestion policy and priority configuration. This specification defines queue modification messages that can be used for queue configuration. Ports are enhanced with scheduling policies and parameters. Port QoS properties may be configured using extensions to the port modification message.

4.2 Packet Flows

4.2.1 VPWS with QoS

The following sections describe the additional tables and actions used for QoS.

4.2.1.1 *Attachment Circuit (UNI->NNI)*

Classification at the UNI can be simple or complicated. Simple classification uses packet QoS fields, including VLAN PCP and IP DSCP. Complicated classification is based on any combination of physical port, VLAN ID, VLAN PCP, VLAN DEI, IP DSCP fields, MAC-SRC, MAC-DST, IP source address, IP destination address, IP Protocol, and optionally, source and destination TCP/UDP ports. The MPLS_TC (EXP) field marking supports EXP-inferred-PSC LSP (E-LSP) for the Pipe model.

Figure 5 shows the additional packet processing objects for QoS highlighted using green coloring.

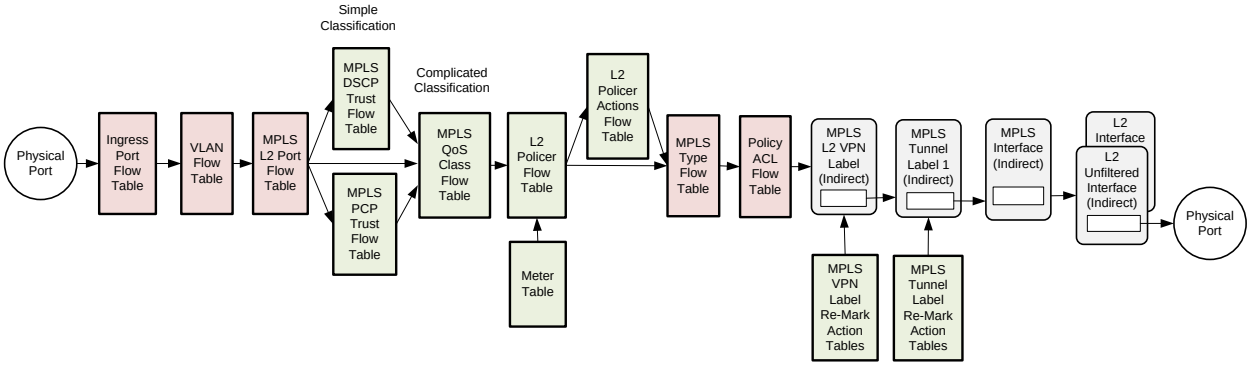

Figure 5 VPWS UNI->NNI Packet Flow with QoS Objects

The following flow tables are introduced:

- **MPLS L2 Port DSCP Trust Flow Table**. An MPLS L2 Port Flow Table rule can include a Goto-Table instruction specifying this table to classify an IPv4 packet by mapping the IP packet DSCP field. This table is used to factor simple classification mapping actions from the MPLS L2 Port Flow Table in order to reduce the number of flow entries. An Apply-Actions instruction is used to set Traffic Class and Color.
- **MPLS L2 Port PCP Trust Flow Table**. This table performs similar simple classification mapping actions for a service flow but matches the VLAN priority field. It also includes an Apply-Actions instruction to set Traffic Class and Color.
- **MPLS L2 Port QoS Class Flow Table**. This table performs complicated classification as described above, and supports wildcard matching to select particular combinations of fields.

Since the VLAN ID and port have already been used to set the MPLS_L2_PORT, it is matched instead of VLAN ID and port. Flow entries have an Apply-Actions instruction to set packet Traffic Class and Color.

- **L2 Policer Flow Table**. Rules in this table match on the TUNNEL_ID and MPLS_L2_PORT pipeline fields to provide per-service meters. These rules have a Meter Instruction capable of changing the packet Color field. The rule also supplies an index for selecting color-based actions as a result of metering. If no metering is required the matching rule Goto-Table instruction specifies the Policy ACL table.
- **L2 Policer Actions Flow Table**. A matching L2 Policer Flow Table entry provides an index into this table for color-based actions, such as remarking the IP DSCP and 802.1Q PRI header fields. It can also change the Traffic Class. A separate table is required after the L2 Policer Flow Table since actions cannot be applied in the same table as the Meter instruction.

The following action objects[6] are introduced:

- **MPLS VPN Label Re-Mark Table**. An action table that can be used to provide a new MPLS_TC value for an MPLS VPN or SWAP shim header based on the outgoing Traffic Class and Color values for the packet. Invoked with a Set-MPLS-TC-From-VPN-Table action in an MPLS VPN Label Group entry action set to remark the MPLS_TC field in the outermost shim header. It can also be invoked with a Set-MPLS-PCPDEI-From-VPN-Table action to remark the PCP value on the outermost VLAN tag in the frame.
- **MPLS Tunnel Label Re-Mark Table**. An action table that can be used to provide a new MPLS_TC value for MPLS LSP shim headers based on the outgoing Traffic Class and Color values for the packet. Invoked with a Set-MPLS-TC-From-Tunnel-Table action in the MPLS Tunnel Label Group entry action set to remark the MPLS_TC field in the outermost MPLS shim header. It can also be invoked with a Set-MPLS-PCPDEI-From-Tunnel-Table action to remark the PCP value on the outermost VLAN tag in the frame.
- **Meter Table**. A Meter instruction includes a Meter Table entry identifier reference. Meters bands in the referenced entry can specify the marker algorithm, e.g., RFC2698 or RFC4115. The packet Color pipeline field may be changed as the result of applying the Meter Table entry.

Queue and port QoS enhancements are also introduced:

- **Queue Enhancements.** Queues are enhanced with properties for specifying a congestion mode option with mode specific parameters.
- **Port Enhancements.** Ports are enhanced with scheduling configuration.

4.2.1.2 *Network Interface (NNI->UNI)*

Classification at the NNI is based on the MPLS_TC field of the currently outermost MPLS shim header. The QoS objects are shown in Figure 6.

[6] Action objects are similar to flow tables but invoked using actions. Meter is an example of an action object, although in OpenFlow 1.3.4 it is invoked using an instruction.

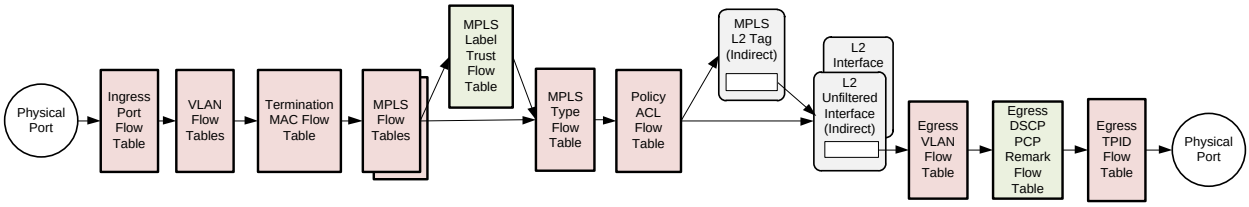

Figure 6 VPWS NNI->UNI Flow with QoS Objects

The following flow tables are introduced:

- **MPLS Label Trust Flow Table.** This table is use to match a copy of the MPLS_TC field value along with an index to set the Traffic Class and Color for the frame after labels have been popped. Rules in the MPLS Flow Tables can use the Copy Field action to make a copy of the MPLS_TC from the label that is to determine QoS properties[7]. The copy is matched from a temporary register.
- **Egress DSCP PCP Remark Flow Table.** This table is optionally used to remark the IP DSCP and VLAN PCP/DEI fields based on the current Traffic Class and Color values. For SPTN this table is primarily used for UNI-UNI interoperability when an SD-Tag is pushed in an MPLS L2 Tag group entry.

This function is not performed in a standard match action table since the matching header field values are no longer available in the frame once the shim header is popped, and there may need to be multiple such shim headers popped. This avoids the complications of defining and passing multiple single purpose pipeline match fields and allows processing of a variable number MPLS shim headers.

A Meter Table entry is also used:

- **Meter Table**. Same as previously described.

Queue and Port QoS extensions are also used:

- **Queue Enhancements.** Queues are enhanced as previously described.
- **Port Enhancements.** Ports are enhanced as previously described.

4.2.2 LSR with QoS

Per-hop behavior is supported for the LSR use case if required using the same tables and actions previously defined for the other use cases. Figure 7 highlights the added QoS objects.

[7] This uses the mechanisms described in the OpenFlow Copy Field extension (EXT-244)

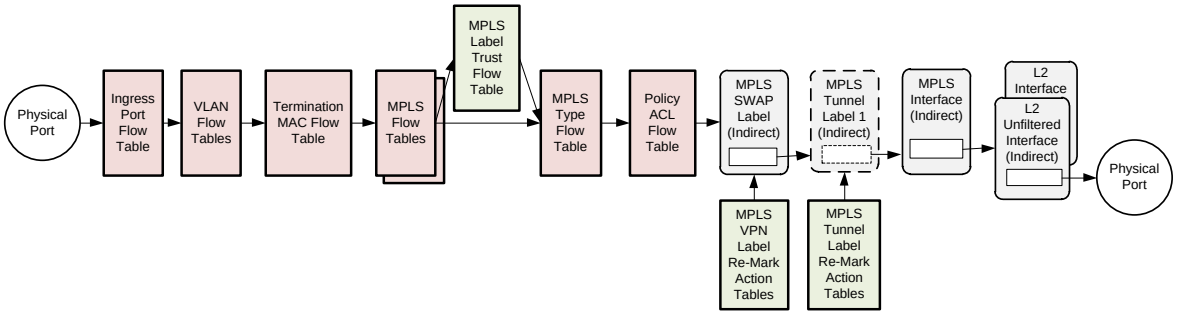

Figure 7 MPLS-TP LSR with QoS Objects

The following actions are used:

- **MPLS Label Trust Flow Table.** Same as previously described.
- **MPLS VPN Label Re-Mark Table**. Same as previously described.
- **MPLS Tunnel Label Re-Mark Table**. Same as previously described.

Queue and Port QoS extensions are also used:

- **Queue Enhancements.** Queues are enhanced as previously described.
- **Port Enhancements.** Ports are enhanced as previously described.

4.3 OpenFlow Extensions for QoS

4.3.1 Match Fields

No additional header match fields are needed to support QoS.

4.3.2 Pipeline Match Fields

The following new pipeline match fields are introduced to support QoS:

- **Traffic Class**. Packet priority. Must take a value between zero and seven, according to the PHB class: BE (0), AF1 (1), AF2 (2), AF3 (3), AF4 (4), EF (5), CS6 (6), and CS7 (7).
- **Color**. Packet drop precedence. Must take a value between zero and two: Green (0), Yellow (1), or Red (2).
- **QoS Index**. Provides an index into a QoS profile entry. Used to reduce the number of entries in the simple classification tables, the MPLS Set QoS action table, and the MPLS Label Re-Mark action tables.
- **Color_Actions_Index**. Provides an index into the L2 Policer Actions Flow Table to select a particular entry.
- **PKT_REG(N).** Packet registers defined in the OpenFlow extensions and used to maintain a temporary copy of a match field. In this specification PKT_REG(0) is used for a copy of the MPLS_TC header field and then used as a match field.

Unless defined in OpenFlow extensions, these require Experimenter protocol definitions.

4.3.3 Actions

The following actions are introduced.

- **Set MPLS TC from Table**. Set the value of the MPLS_TC (EXP) field in the MPLS shim header based on an MPLS VPN Label Re-Mark Action or an MPLS Tunnel Label Re-Mark Action table entry, depending on the group entry type from which it is invoked. The entry is referenced up by QoS Index, Traffic Class, and Color. The flow entry invokes the action with the QoS Index. Traffic Class and Color parameters are supplied by the switch.
- **Set PCP DEI From Table**. Set the value of the priority and drop indicator fields the outermost VLAN tag based on an MPLS VPN Label Re-Mark Action or an MPLS Tunnel Label Re-Mark Action table entry, depending on the group entry type from which it is invoked. The entry is referenced by QoS Index Traffic Class, and Color. The flow entry invokes the action with the QoS Index. Traffic Class and Color parameters are supplied by the switch.
- **Copy Field**. Copy a value from one field to another. Used here with source MPLS_TC header field and destination PKT_REG(0).

These require Experimenter protocol definitions.

4.3.4 Objects

The following objects are introduced in conjunction with the actions described in the previous section.

- **MPLS VPN Label Re-Mark Table**. Remark table action objects have been previously described for use with MPLS VPN or MPLS SWAP group entries.
- **MPLS Tunnel Label Re-Mark Table**. Remark table action objects have been previously described for use with MPLS Tunnel 1 group entries.

Unless defined in OpenFlow extensions, these require Experimenter protocol definitions as well as new Experimenter message types for modifying and querying entries.

4.3.5 Ports and Queues

Ports and queues are enhanced:

- **Ports.** In the OpenFlow 1.3.4 specification scheduling is left as a local matter. This specification enhances ports with explicit scheduling capabilities.
- **Queues**. Queue configuration is outside of the OpenFlow 1.3.4 specification. This specification adds message types for queue configuration. In addition, it adds configuration options for specifying scheduling and congestion properties.

These require Experimenter extensions, as well as new Experimenter message types for modifying and querying queues. Configuration extensions are defined for port scheduling properties.

4.3.6 **Color Set Meter Bands**

SPTN Meter Table entries contain the two color set meter bands described in Section 4.1. These differ from the existing OpenFlow 1.3.4 bands in that they apply an action to set the Color packet metadata field.

Different band types are used to distinguish between color-aware and color-blind operation. A meter entry can only contain two Color Set meter bands. Both must be either color-aware or color blind, different types cannot be mixed in the same meter entry. Color Set meter bands cannot be mixed with existing drop or DSCP remark meter bands.

An experimenter definition is used for this. The new color is specified when programming the band and can only take values Yellow (1) or Red (2).

The meter can access the current value of the Color pipeline match field to support color-aware operation.

4.4 Configuration Extensions

Port scheduling mode is a port property that is configured for a port using extensions to OF-CONFIG. There can be one scheduling mode for a port for all queues on that port. Scheduling mode can be one of: Strict Priority (SP); Round Robin (RR); Weighted Round Robin (WRR); Weighted Deficit Round Robin (WDRR), and combined modes SP+WRR and SP+WDRR. The default is strict priority (SP) mode.

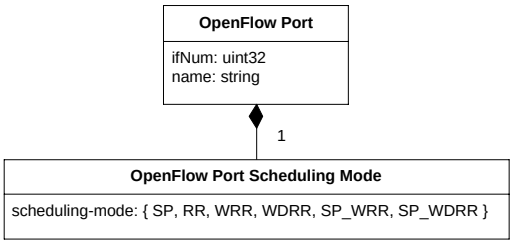

Figure 8 Port Scheduling Mode Properties

For priority and weighted modes the priority and/or weights must be configured on each queue. Port and queue scheduling properties must be properly configured for predictable scheduling behavior. For WFQ use WRR and set the scheduling weights for all queues equal.

Queue scheduling and congestion properties can be configured using either Experimenter OpenFlow messages,[8] described in Section 8, or OF-CONFIG queue configuration extensions. Figure 9 shows the OpenFlow Queue Property configuration extensions to OpenFlow Queue objects.

[8] This re-introduces the capability that was in OpenFlow 1.0

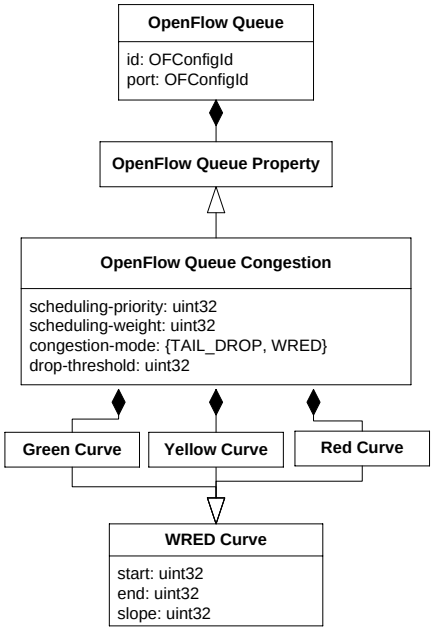

Figure 9 Queue Configuration Properties

The congestion mode can be either Tail Drop or Weighted Random Early Discard (WRED), Tail Drop being the default. Tail Drop can be optionally configured with a relative drop-threshold. Figure 10 illustrates WRED curve configuration settings.

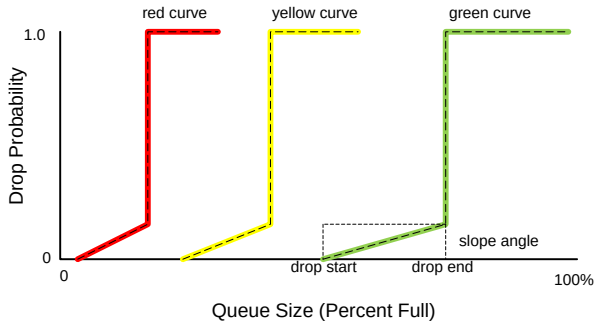

Figure 10 WRED Configuration

5 OAM

This section describes the SPTN logical switch objects supporting OAM. OAM has two aspects: defining the Maintenance Points in terms of flow table matches and actions, and configuring the OAM PDU processing functions performed by a local OAM engine. This section discusses OAM concepts and requirements followed by the packet flow diagrams.

5.1 OAM Concepts

5.1.1 Overall Requirements

OAM functions are required at the MPLS-TP and Ethernet Service layers. MPLS-TP PW, and LSP follow ITU-T Recommendation G.8113.1/Y.1372.1. Ethernet service OAM follows IEEE 802.1ag and ITU-T Recommendation G.8013/Y.1731. Ethernet link OAM follows IEEE 802.3ah[9]. Most (but not all) OAM scenarios are illustrated in Figure 11.

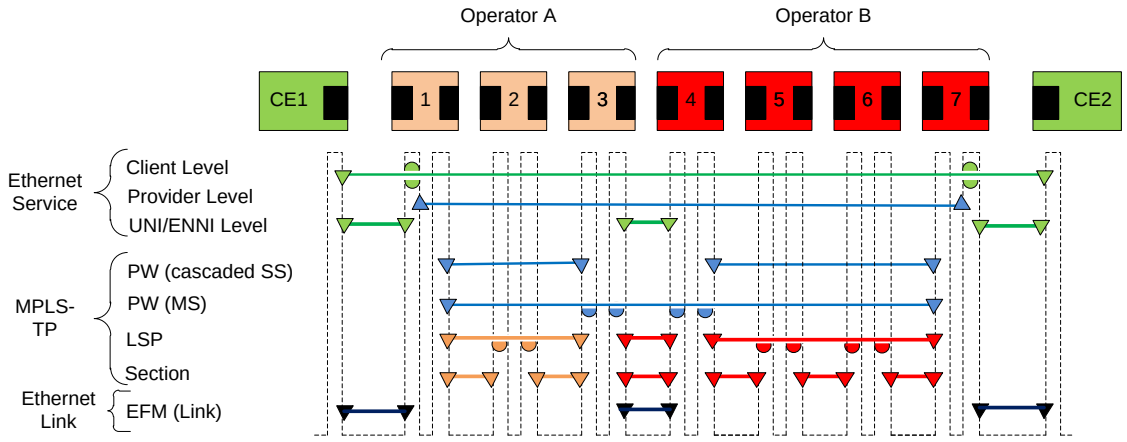

Figure 11 MPLS-TP Service OAM Scenarios

OAM feature support by layer is shown in Table 1.

Table 1 OAM Function Support

Type		Function	Section OAM	LSP OAM	PW OAM	Service OAM	Link OAM
Proactive	Fault Management	CCM	Yes	Yes	Yes	Yes	Yes
		AIS		Yes		Yes	
		RDI	Yes	Yes	Yes	Yes	
		LCK		Yes	Yes		
		CSF			Yes		
	Performance Monitoring	Packet Loss Measurement by CCM		Yes			
On-demand	Fault Management	LB	Yes	Yes	Yes	Yes	Yes
		LT		Yes		Yes	
		TST		Yes	Yes		
	Performance Monitoring	LM	Yes	Yes	Yes	Yes	
		DM	Yes	Yes	Yes	Yes	

[9] EFM is included for reference purposes but not specified in this version.

Note that the loss measurement function referred in this specification is applicable to both CCM-based proactive LM and LMM/LMR based on-demand/proactive LM.

5.1.2 OAM PDU Local Processing Model

OAM PDU processing is done in an engine external to but co-located with the OFLS to meet performance requirements. This is modeled using a local processing function. The interface with the OFLS uses the LOCAL reserved port and permits pipeline match fields to accompany a packet that is output to LOCAL. The processor uses the LOCAL reserved port to inject packets along with pipeline match fields into the head of the pipeline.

This relationship is diagramed in Figure 12.

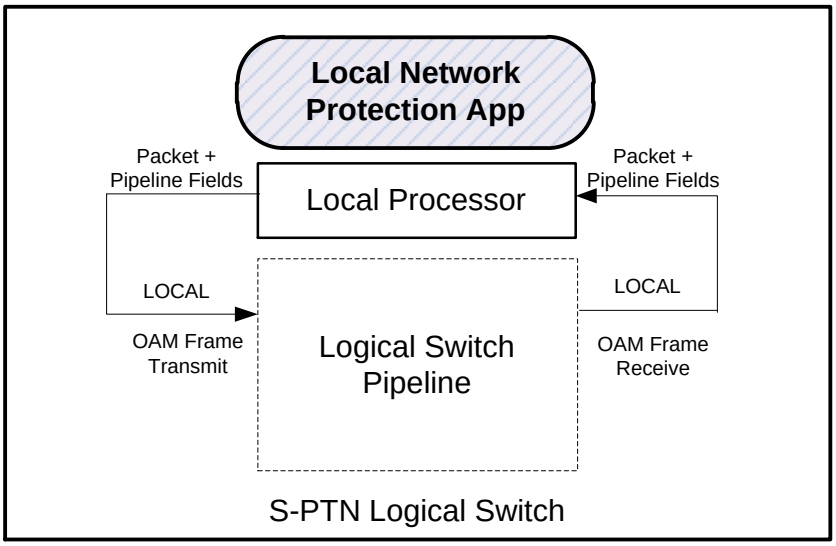

Figure 12 OAM Processing Model

One particular pipeline match field, openFlowMpId, must always accompany a packet between the OFLS and the local processing function. This field links the protocol pipeline processing programmed by OpenFlow with the local processing function programmed using OF-CONFIG.

- The openFlowMpId field must be set by rule action for packets output to the LOCAL reserved port
- Frames are received from the LOCAL reserved port with a non-zero openFlowMpId field initialized by the local processing function
- Frames received from ports other than the LOCAL reserved port have a openFlowMpId field initialized to zero.

5.2 Packet Flows

The following sections describe the additional tables and actions used to support OAM.

5.2.1 Ethernet Service OAM

Client Service OAM is end-end between client points of service (UNI-C). Since MPLS-TP provides an Ethernet service, Client Service OAM exchanges pass through the MPLS tunnel

between the MEPs on customer equipment (CE). For Client Service OAM the provider installs MIPs on provider edge equipment (PE).

Provider Service OAM is end-end between service provider points of service (UNI-N). Provider Service OAM exchanges pass through the MPLS tunnel between PE Up MEPs.

UNI/ENNI OAM is on the links between client and service or between operator networks.

In general, Up MEP or Up MHF processing for Tx frames (i.e., traffic in the direction of the partner maintenance point) is done in the ingress tables, and Up MEP or Up MHF processing for Rx frames (i.e., traffic from the partner maintenance point) is done in the egress tables. Conversely, Down MEP or Down MHF processing for Rx frames is done in the ingress tables, and Down MEP or Down MHF processing for Tx frames in the egress tables.

5.2.1.1 *OAM Control Frames*

Ethernet Service OAM frames are distinguished by Ethertype. Frames of interest are distinguished further by MDL, and in some cases by opcode and destination MAC address. OAM PDUs that require processing are trapped to the OAM engine.

As before the additional functionality is shown colored in green.

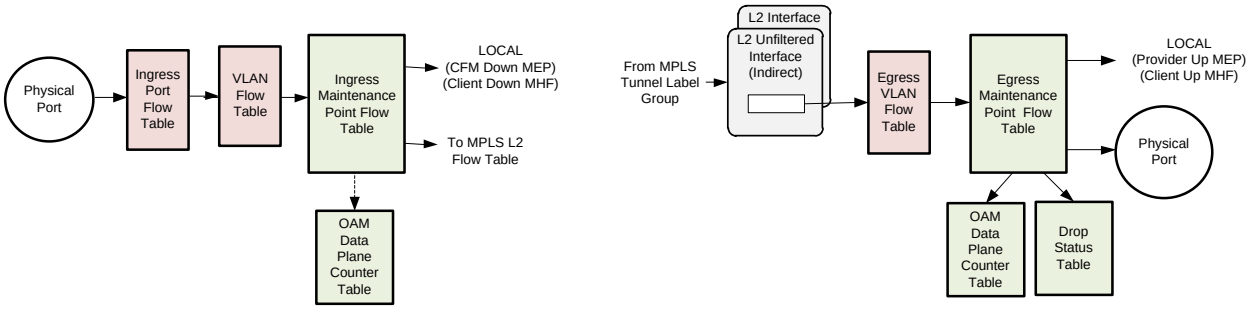

Figure 13 Ethernet Service OAM PDU Packet Processing

The following flow tables are used:

- **VLAN Flow Table.** Flow entries in this table forward any frame associated with a maintenance point to the Ingress Maintenance Point Table. Other frames are forwarded to the MPLS L2 Port Table.
- **Ingress Maintenance Point Flow Table**. Flow entries in this table implement Client Down MHF, UNI Down MEP, and Provider Up MEP processing actions.

  The OAM engine must be able to process Loopback and Link Trace messages for a client Down MHF. OAM PDU flow entries match on Ethertype, MDL and opcode, and have actions to output frames to LOCAL. If Provider Up MEP loss measurement is configured, higher MDL OAM PDUs are counted as data using the OAM Data Plane Counter Table.

  The OAM Data Plane Counter Table maintains counts by direction, openFlowMpId, and traffic class. For LM counting the flow entry action set includes OAM LM Tx and/or Rx Count actions for which an openFlowMpId argument must be provided. The counters are qualified by the packet Traffic Class pipeline match field, and packets are counted when the

action set is processed so that it uses the final packet Traffic Class after all classification, and so that the counter is not incremented for dropped frames due to policing actions.

- **Egress VLAN Flow Table**. Flow entries in this table forward frames associated with a maintenance point functions to the Egress Maintenance Point Flow Table. Egress tables are an OpenFlow extension feature described in Section 5.3.1. Other packets are output.
- **Egress Maintenance Point Flow Table**. Flow entries in this table implement UNI Down MEP and Provider Up MEP processing actions. For OAM PDUs flow entries match on Ethertype, MDL and opcode, and have actions to output OAM frames to LOCAL along with LM counters and arrival timestamp, if applicable.

  Actions are provided to sample transmit and receive LM counter values and set pipeline match field values that will accompany OAM PDUs sent to the OAM engine. For the sampling operation the switch uses the packet Traffic Class. A pipeline match field is also initialized with the frame arrival timestamp for DM.

  The Drop Status Table is an action table object used to prevent frames from being forwarded during a Lock condition. It can be invoked from the flow entry action list using the Check Drop Status action. Packet drop status is read only to the OFLS. Whether or not to drop is controlled by the local OAM processing function. The drop action takes effect immediately.

Implementations may choose to set counter and timestamp fields in OAM PDUs before forwarding to the OAM processing engine. This would be implementation dependent and not programmed using OpenFlow.

The following new match fields are introduced:

- **OAM_Y1731_OPCODE**. This matches the opcode field in the Y.1731 PDU. Prerequisite is packet Ethertype=0x8092. This field is used by the Maintenance Point Flow Tables to determine disposition of the PDU.
- **OAM_Y1731_MDL**. This matches the maintenance domain level field in the Y.1731 PDU. Prerequisite is packet Ethertype=0x8092. This field is used by the Maintenance Point Flow Tables to determine disposition of the PDU.

The following objects are introduced:

- **OAM Data Plane Counter Table**. This table provides loss measurement counters for performance monitoring. It is usually invoked as an action in an action set but can be used in an action list.
- **Drop Status Table**. This table is used to immediately drop frames. It can be invoked as an action with a openFlowMpId argument. This feature is intended to be used when a MEP is in the Locked Administrative State to selectively drop frames.

The following pipeline match fields are used to convey LM and DM values to the OAM engine:

- **TxFCl**. The value of the transmit LM counter for a particular openFlowMpId and Traffic Class. Used for computing Frame Loss Ratio.

- **RxFCl**. The value of a receive LM counter for a particular openFlowMpId and Traffic Class. Used for computing Frame Loss Ratio.
- **RxTime.** Receive timestamp of OAM PDU, used for computing Frame Delay. The value is supplied by the switch.

The following actions are introduced:

- **OAM LM Tx Count**. This is accessed with openFlowMpId and Traffic Class to increment the transmit direction LM counters.
- **OAM LM Rx Count**. This is accessed with openFlowMpId and Traffic Class to increment the receive direction LM counters.
- **Set TxCFl and RxCFl Fields**. This action, used for MEP processing, reads the counter values from the referenced entry in the OAM Data Plane Counter Table and sets counter pipeline match fields. It also sets the receive timestamp. These fields accompany the packet for processing by the OAM engine.
- **Check Drop Status**. This is accessed with an openFlowMpId to check if the MEP is in the Locked state and drop the packet.

5.2.1.2 *Data Frames*

Two functions are required for data frames, incrementing LM counters and enforcing Lock state.

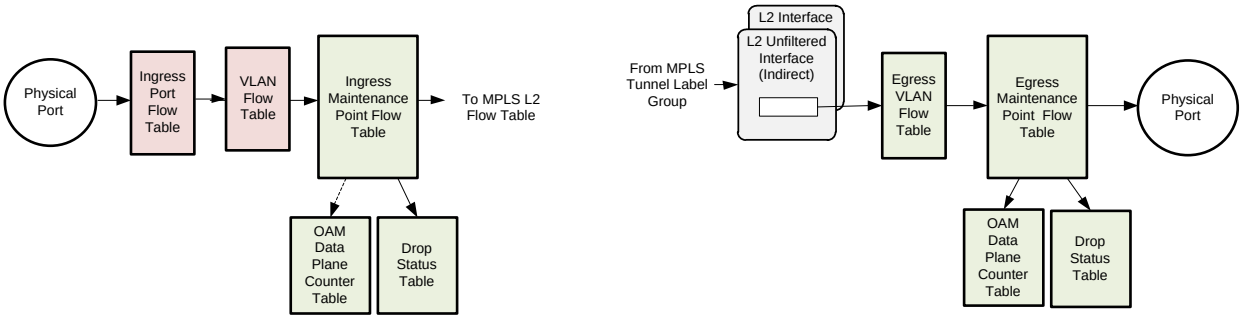

Figure 14 Ethernet Service OAM Data Frame Packet Processing

The following flow tables are involved:

- **VLAN Flow Table.** Flow entries in this table are used to identify data frames that are associated with an Ethernet maintenance point and forward these to the Ingress Maintenance Point Flow Table.
- **Ingress Maintenance Point Flow Table**. This table performs similar functions to those described above except for data frames. If a Provider Up MEP has loss measurement performance monitoring configured, the flow entry includes an action in the action set to increment the Tx count for the openFlowMpId. Data frames are forwarded to the MPLS L2 Port Flow Table unless the MEP is in the administratively Locked state. If LCK processing is configured for a Provider Up MEP, the flow entry should include an action to check the Drop Status Table for the openFlowMpId.

- **Egress VLAN Flow Table**. Flow entries in this table are used to identify data frames that are associated with an Ethernet maintenance point and forward these to the Egress Maintenance Point Flow Table.
- **Egress Maintenance Point Flow Table**. Flow entries in this table process Rx data frames for Provider Up MEPs. These flow entries can have actions to update loss measurement counters before outputting the packet. The Drop Status table is used to implement LCK processing .

The following objects are used:

- **OAM Data Plane Counter Table**. Same as previously described, updated when Action Set is executed after Egress processing and the packet is sent to the output port.
- **Drop Status Table**. Same as previously described.

The following actions are used:

- **OAM LM Tx Count**. This is invoked with openFlowMpId and Traffic Class to increment the transmit LM counters for a MEP. If the Traffic class is not specified it is provided by the switch.
- **OAM LM Rx Count**. This is invoked with openFlowMpId and Traffic Class to increment the receive LM counters for a MEP. If the Traffic class is not specified it is provided by the switch.
- **Check Drop Status.** Same as previously described.

5.2.1.3 *Injected OAM Frames*

For Provider Service and Client UNI OAM, the Local Processor injects PDU frames into the pipeline accompanied by an openFlowMpId, Traffic Class, and Color pipeline metadata fields. For normal frames openFlowMpId must be programmed zero. Figure 15 illustrates the processing flow for these frames.

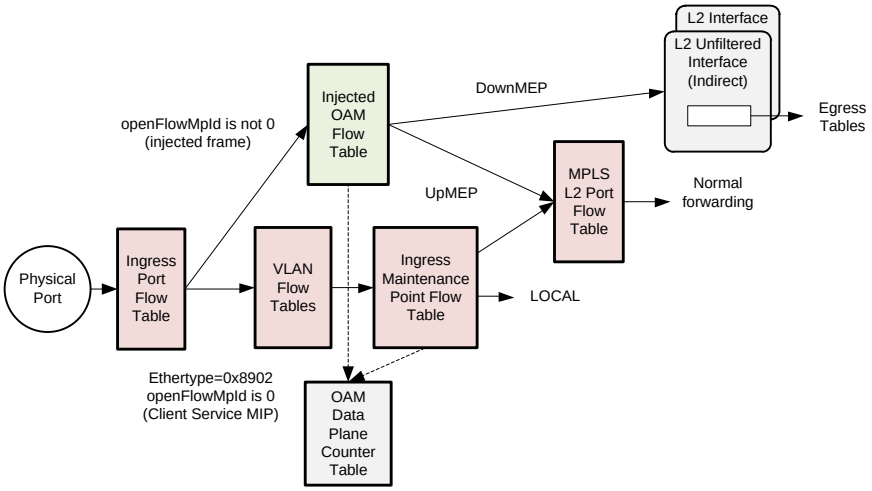

Figure 15 Ethernet OAM PDU Injection

The following new flow table is introduced:

- **Injected OAM Flow Table.** Rules in the Ingress Port Flow Table that match non-zero openFlowMpId values have Goto-Table instructions specifying the Injected OAM Flow Table. This table matches openFlowMpId, Traffic Class, and Color to determine how to encapsulate and forward the packet. For Ethernet OAM frames a matching rule assigns MPLS_L2_PORT and TUNNEL_ID pipeline fields for a particular service. From this point on the frame is forwarded as though a customer service frame.

The following objects are used:

- **OAM Data Plane Counter Table**. Same as previously described, updated when Action Set is executed after Egress processing.

The following actions are used:

- **OAM LM Tx Count**. This is invoked with openFlowMpId and Traffic Class to increment the transmit LM counters for a MEP. The packet Traffic Class is provided by the switch when the action is executed.
- **OAM LM Rx Count**. As previously described.

5.2.2 MPLS-TP OAM

5.2.2.1 *Control Frames*

For a Down MEP Control Frame actions are associated with LSP and PW label processing.

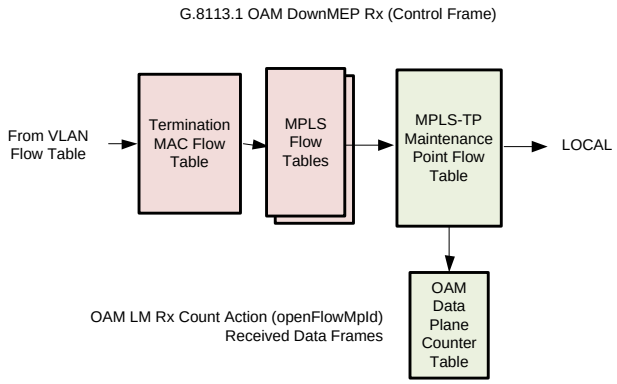

Figure 16 VPWS NNI->UNI Packet Flow with OAM Objects (Control Frame)

The following new flow tables are introduced:

- **MPLS Flow Tables.** Flow entries in this table are used to identify control frames that are associated with a Down MEP. The MPLS Flow Table entry matches on fields (described below) to identify control frames and then apply actions to pop the labels and associated fields and set the openFlowMpId pipeline match field. These actions logically transform the packet into a Y.1731 PDU frame. These rules have a Goto-Table instruction specifying the MPLS-TP Maintenance Point Flow Table.

  MPLS-TP Maintenance Point Flow Table. This table receives Y.1731 frames from the MPLS Flow Table. Flow entries in this table match on MDL and opcode with actions to output frames to LOCAL based on opcode. It performs a similar function as the Ethernet Maintenance Point Flow table described earlier.

Implementations may choose to update counter and timestamp fields in OAM PDUs before forwarding. This functionality would be built-in and not programmed using OpenFlow.

- **OAM Data Plane Counter Table.** This action table is used to count frames for Performance Monitoring Loss Measurement if needed. The flow entry action set can include the OAM LM Rx Count action specifying the openFlowMpId. As before, the Traffic Class argument is implicitly provided by the switch.

The following match fields are used:

- **MPLS_DATA_FIRST NIBBLE.** Matches the four bits immediately following the bottom of stack label to determine if a CW (0) or ACH (1).
- **MPLS_ACH_CHANNEL.** This matches the channel field in an ACH header. The prerequisite is an MPLS frame with MPLS_DATA_FIRST_NIBBLE = 1. The value for MPLS-TP OAM should match 0x8902. However 0x7FFA could also be programmed for backward compatibility.
- **MPLS_NEXT_LABEL_IS_GAL.** This is set by the parser for MPLS frames. This is used to identify certain OAM formats without requiring an additional lookup.
- **OAM_Y1731_OPCODE**. Same as previously described.
- **OAM_Y1731_MDL**. Same as previously described. The only value permitted is 7.

The following stateful function objects are used:

- **OAM Data Plane Counter Table**. Same as previously described.

The following pipeline match fields are used to convey LM and DM values to the OAM engine:

- **TxFCl**. Same as previously described.
- **RxFCl**. Same as previously described.
- **RxTime.** Same as previously described.

The following actions are used:

- **OAM LM Rx Count**. Same as previously described.
- **Set TxCFl and Rx CFl Fields**. As above, this action sets pipeline match fields from entries in the OAM Data Plane Counter Table for a MEP. These fields accompany the packet for processing by the OAM engine.

5.2.2.2 *Data Frames*

MPLS TP-OAM associates MIP and Down MEP rules with LSPs and PWs.

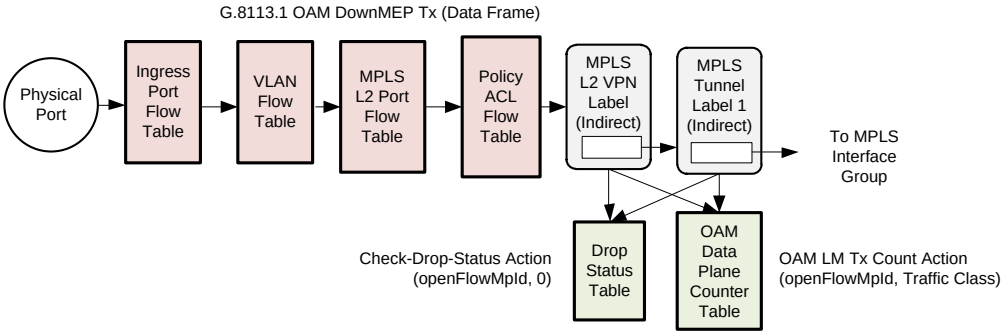

Figure 17 VPWS UNI->NNI Packet Flow with OAM Objects (Data Frame)

The following flow tables and group entries are involved:

- **MPLS L2 VPN Label, MPLS Tunnel Label 1**. These groups are augmented with actions to increment transmit counters for the MPLS Down MEP. The action set includes the OAM LM Tx Count action specifying the openFlowMpId. The Traffic Class argument is provided by the switch. The group entries can check the locked administrative state of an MPLS MEP associated with the PW or LSP using the same Check Drop Status action as for Ethernet OAM.

The following action and counter objects are used:

- **OAM Data Plane Counter Table**. Same as previously described.
- **Drop Status Table**. Same as previously described.

The following actions are used:

- **OAM LM Tx Count**. This is accessed with MEP ID and Traffic Class from a group action set to increment the transmit LM counters for a MEP. The Traffic Class is provided by the switch.
- **Check Drop Status**. Same as previously described.

MPLS TP-OAM MIP and Down MEP rules are associated with LSP and PW label processing.

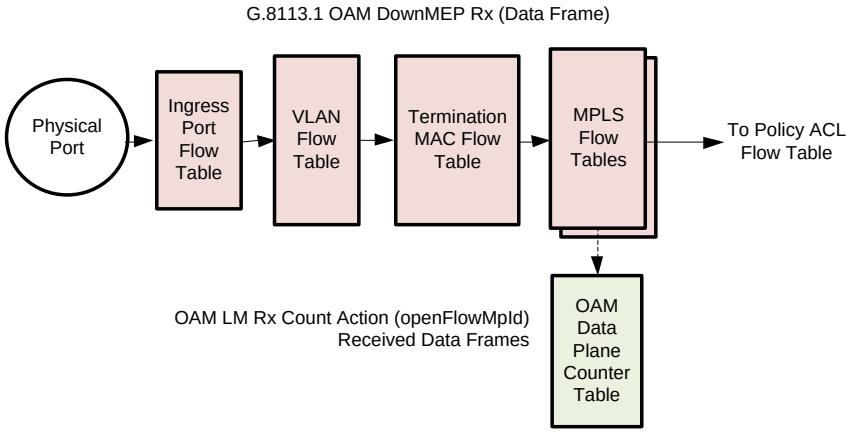

Figure 18 VPWS NNI->UNI Packet Flow with OAM Objects (Data Frame)

The following flow tables are involved:

- **MPLS Flow Tables.** Flow entries in this table are used to identify incoming data frames that are associated a Down MEP. If the MEP has loss measurement performance monitoring configured the flow entry adds an action in the action set to increment the Rx LM count for the openFlowMpId. The count action is in the action set so that it is deferred until after traffic conditioning and security.

  The flow entry can check the locked administrative state of a Down MEP associated with the flow using the Check Drop Status action.

The following action objects are used:

- **OAM Data Plane Counter Table**. Same as previously described.

The following actions are used:

- **OAM LM Rx Count**. Same as previously described.

5.2.2.3 *Injected OAM Frames*

Similar to Ethernet, the Local Processor can inject G.8113.1 OAM PDU frames with Traffic Class and Color pipeline fields. These frames are distinguished by an accompanying non-zero openFlowMpId pipeline field that determines how these frames are encapsulated. For normal frames this field would not be present and defaults to zero. Figure 19 illustrates processing of these frames.

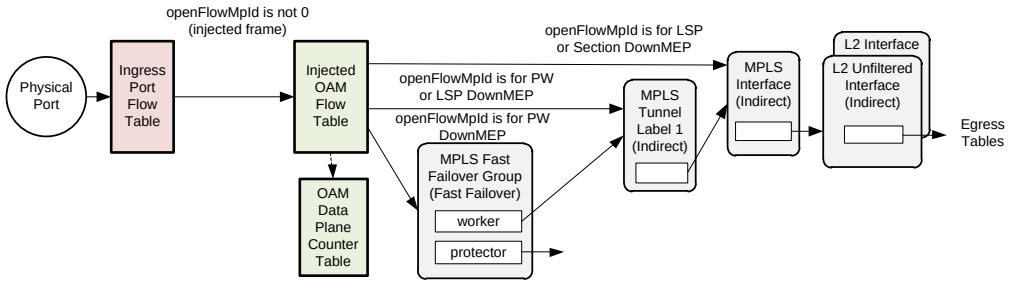

Figure 19 G.8113.1 OAM PDU Injection

The following flow table is used:

- **Injected OAM Flow Table.** Rules in the Ingress Port Flow Table that match non-zero openFlowMpId values have Goto-Table instructions specifying the Injected OAM Flow Table. This table matches openFlowMpId, Traffic Class, and Color, and forwards the frame into the appropriate location in the pipeline.

  Down MEP OAM Frames are injected as Ethernet (Y.1731). By matching the openFlowMpId, these frames can be assigned associated pipeline header field values and edited for forwarding.

  For PW OAM an MPLS PW label is pushed with an Associated Channel header (ACh) and other header fields depending on the VCCV type, and then the frame is forwarded to an MPLS Tunnel Label group. For LSP OAM an MPLS GAL label is pushed along with an

ACh shim header followed by push of an LSP label and forwarding to an MPLS Interface group entry.

Received OAM frames for the Down MEPs are handled as previously described for Figure 16.

5.2.3 **LSR OAM**

The LSR OAM case is shown as in Figure 20. The MPLS Flow Tables use an action in the action set to increment the receive LM counter for the MEP. Note that this action is not performed if the packet is dropped due to a policing action.

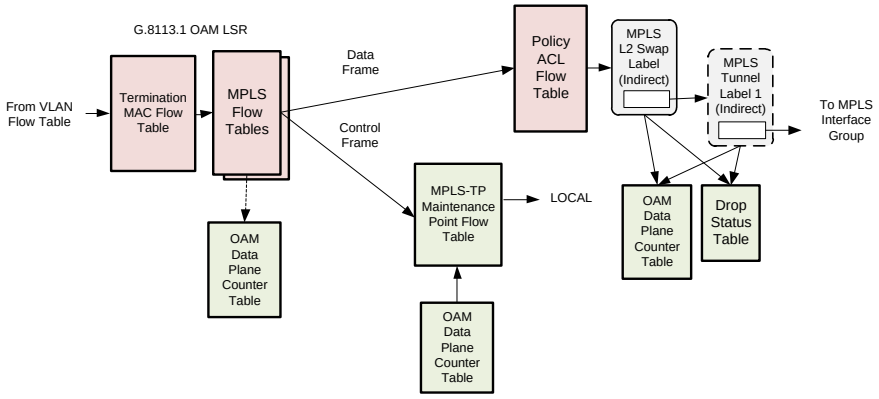

Figure 20 MPLS-TP LSR with OAM Objects

The following flow tables are involved:

- **MPLS Flow Tables.** Same as previously described.
- **MPLS-TP Maintenance Point Flow Table**. Same as previously described.

The following objects are used:

- **OAM Data Plane Counter Table**. Same as previously described.
- **Drop Status Table**. Same as previously described.

The following pipeline match fields are used to convey LM and DM values to the OAM engine:

- **TxFCl**. Same as previously described.
- **RxFCl**. Same as previously described.
- **RxTime.** Same as previously described.

The following actions are used:

- **OAM LM Rx Count**. Same as previously described.
- **OAM LM Tx Count**. Same as previously described.
- **Set TxCFl and Rx CFl Fields**. Same as previously described.
- **Check Drop Status**. Same as previously described. Used for both control and data frames.

5.3 OpenFlow Extensions

A number of extensions to OpenFlow are used to support OAM. These are all programmed using experimenter protocol fields. These extensions are described in this section.

5.3.1 Egress Flow Tables

Egress flow tables constitute a restricted form of match action table that are used once all forwarding is done for a packet but immediately before it is output. Egress tables always process packets at the output port. Egress tables cannot change the output port nor specify a group action, but can mirror or drop a packet.

In this specification the egress table pipeline can consist of the Egress VLAN Flow Table, the Egress Maintenance Point Flow Table, the Egress DSCP PCP Remark Flow Table, and the Egress TPID Flow Table.

Egress tables are introduced in OpenFlow 1.5 but with minimal protocol impact. OpenFlow 1.3.4 provides basic protocol support for egress tables as long as the constraints identified here are followed and the first egress table number is fixed. For this specification the first egress table is always the Egress VLAN Flow Table.

Egress tables effectively partition the pipeline, and the table numbering space is partitioned accordingly. Egress table Goto-Table instructions can only specify higher numbered egress tables. A lower numbered table than the first egress table cannot have a Goto-Table instruction that specifies an egress table.

Egress table flow entries cannot change the packet output port action. The packet output action remains in place during egress pipeline processing and is taken after the last egress table, assuming the packet is not dropped.

5.3.2 Match Fields

The following header match fields were previously defined:

- **MPLS_DATA_FIRST NIBBLE**
- **MPLS_ACH_CHANNEL**
- **OAM_Y1731_OPCODE**
- **OAM_Y1731_MDL**

5.3.3 Pipeline Match Fields

The following pipeline metadata fields are defined:

- **MPLS_NEXT_LABEL_IS_GAL.** This is set by the parser for MPLS frames that contain a GAL(13) label immediately after an LSP or PW label. This is used to identify certain OAM formats without requiring an additional table lookup.
- **ACTSET_OUTPUT**. Permits an egress table to match on the output port.[10]

[10] This field is encoded consistent with the definition in OpenFlow 1.5.

- **openFlowMpId**. Used to identify the MEP or MIP that should process a received PDU, or the destination MEP or MIP for an injected frame.
- **TxCFl**. OAM Data Counter field for the transmitted frames LM counter for a particular MIP. This is read by a Set TxCFl and RxCFl action for accompanying a control PDU Packet_In.
- **RxCFl**. OAM Data Counter field for the received frames LM counter for a particular MIP. This is read by a Set TxCFl and RxCFl action for accompanying a control PDU Packet_In.
- **RxTime**. Arrival timestamp set by the hardware. This field accompanies a control PDU Packet_In.

5.3.4 Action Objects

The following new objects are invoked as stateful actions:

- **OAM Data Plane Counter Table**. This table maintains loss measurement counters for performance monitoring. It is invoked as an action in either an action set or action list. It is defined as a new object primarily because it selectively provides an aggregated frame count for particular maintenance points. Another reason is since it may be invoked from multiple points in the pipeline, such as from flow table or group entry action sets.
- **Drop Status Table**. This table is used to selectively prevent forwarding frames. It is invoked in an action set with a openFlowMpId argument. If the MEP referenced by the openFlowMpId is in the Locked Administrative State the packet is dropped. The local OAM process is externally configured with the MEP administrative state and sets or clears the drop indication accordingly.

5.3.5 Actions

The following actions are defined:

- **OAM LM Rx Count**. Increment the LM Rx counter value in the OAM Data Plane Counter Table for a specified MEP ID. The Traffic Class is supplied by the switch.
- **OAM LM Tx Count.** Increment the LM Rx counter value in the OAM Data Plane Counter Table for a specified MEP ID. The Traffic Class is supplied by the switch.
- **Set TxCFl and Rx CFl Fields**. This action sets counter pipeline match fields from entries in the OAM Data Plane Counter Table. It also sets the arrival timestamp. These fields accompany the packet for processing by the OAM engine.
- **Check Drop Status**. Drop packet if there is a drop entry for a particular MEP openFlowMpId.

5.4 Configuration Extensions

Configuration extensions are used to set parameters for OAM processing in a standard way.

These extensions are described in a generic way using information models. These information models follow the ONF modeling guidelines.

5.4.1 Ethernet Service OAM

5.4.1.1 *Fault Management*

Figure 21 shows the information model for configuring Ethernet Service OAM Fault Management. Note that the Ethernet_MEG class is an OF-CONFIG resource instance.

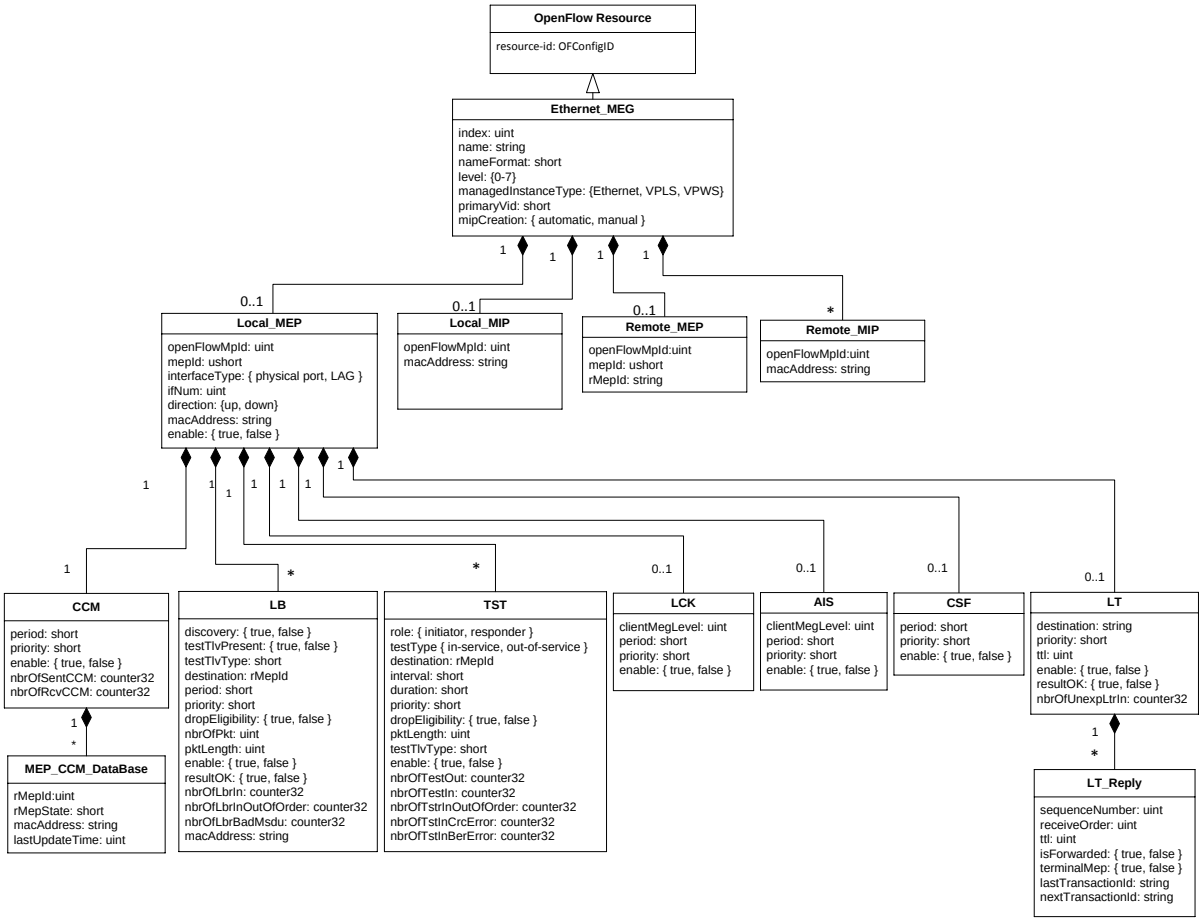

Figure 21 Ethernet Service OAM Fault Management

At any particular maintenance level the node can be either a MEP or a MIP. For a MEP level node, Local_MEP and Remote_MEP objects must be configured. One or more Remote_MIPs can optionally be configured. At a MIP level node only a Local_MIP would be configured.

5.4.1.2 *Performance Monitoring*

Figure 22 shows the configuration information model for Ethernet Service OAM Performance Monitoring.

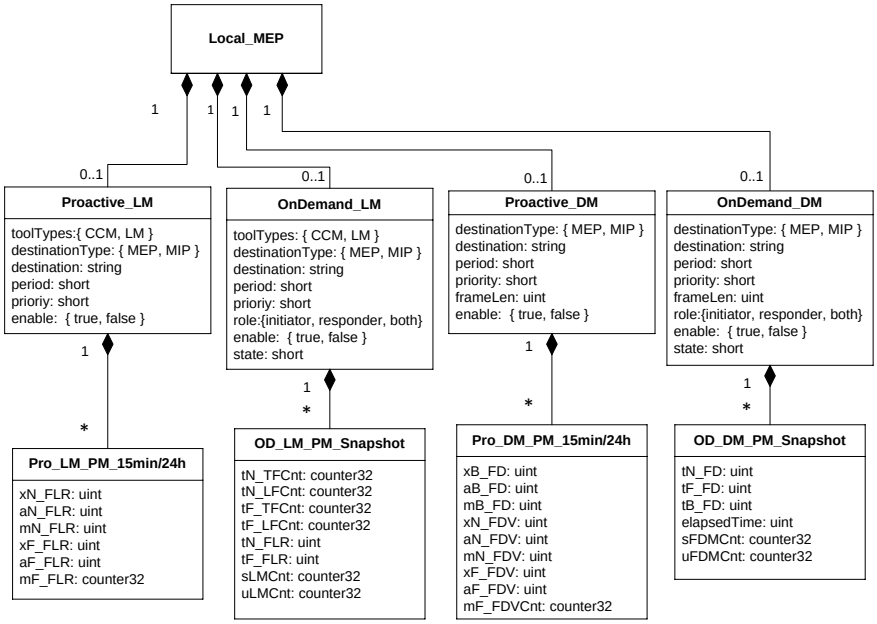

Figure 22 Ethernet Service OAM Performance Monitoring

5.4.2 **MPLS-TP OAM**

5.4.2.1 ***Fault Management***

Figure 23 shows the configuration information model for MPLS-TP OAM Fault Management. As was the case with the Ethernet-MEG, the G.8113.1_MEG is also an instance of an OF-CONFIG resource.

Note that the Local_MEP class has a property indicating its server layer. This is used when the managedInstanceType property is PW or LSP.

At any particular layer the node can be either a MEP or a MIP. For a MEP level node, Local_MEP and Remote_MEP objects must be configured, and one or more Remote_MIPs can optionally be configured. At a MIP layer node only a Local_MIP would be configured.

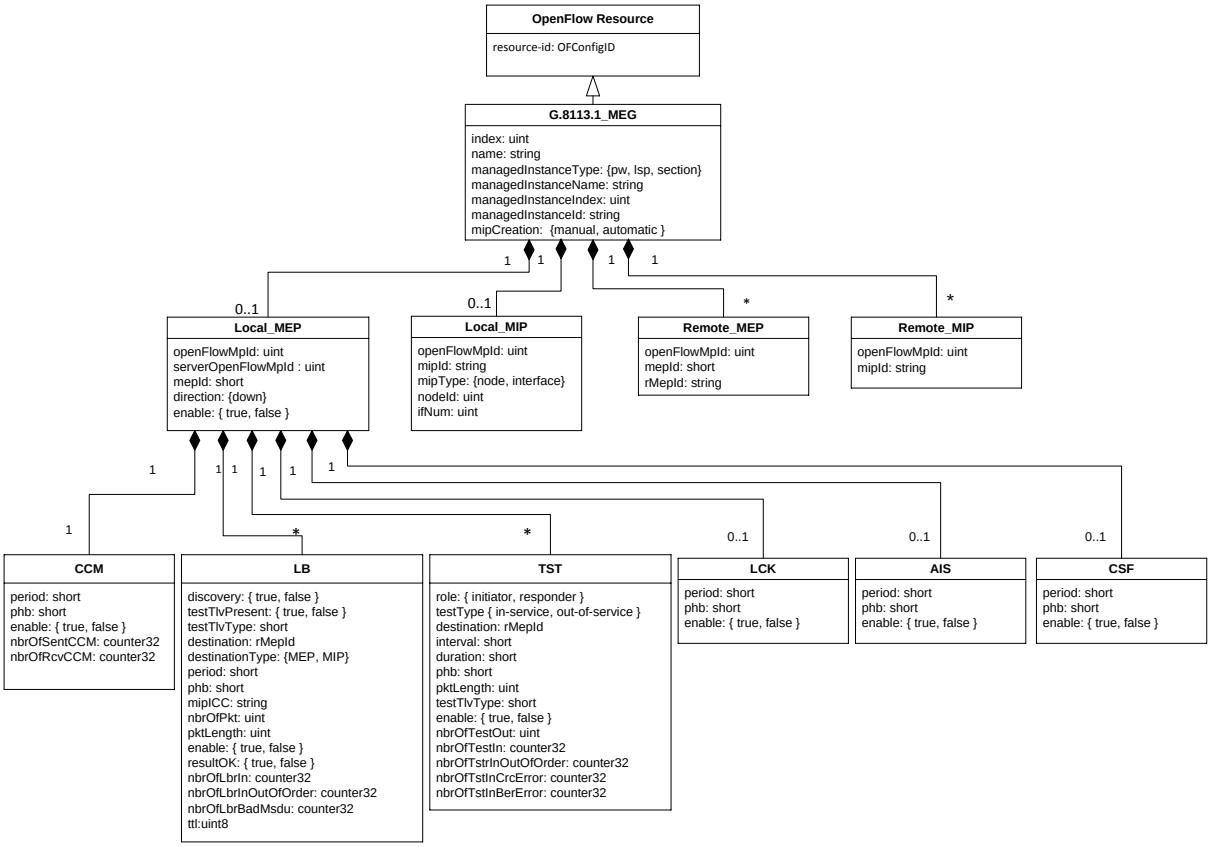

Figure 23 MPLS-TP OAM Fault Management

5.4.2.2 ***Performance Monitoring***

Figure 24 shows the configuration information model for MPLS-TP Performance Monitoring.

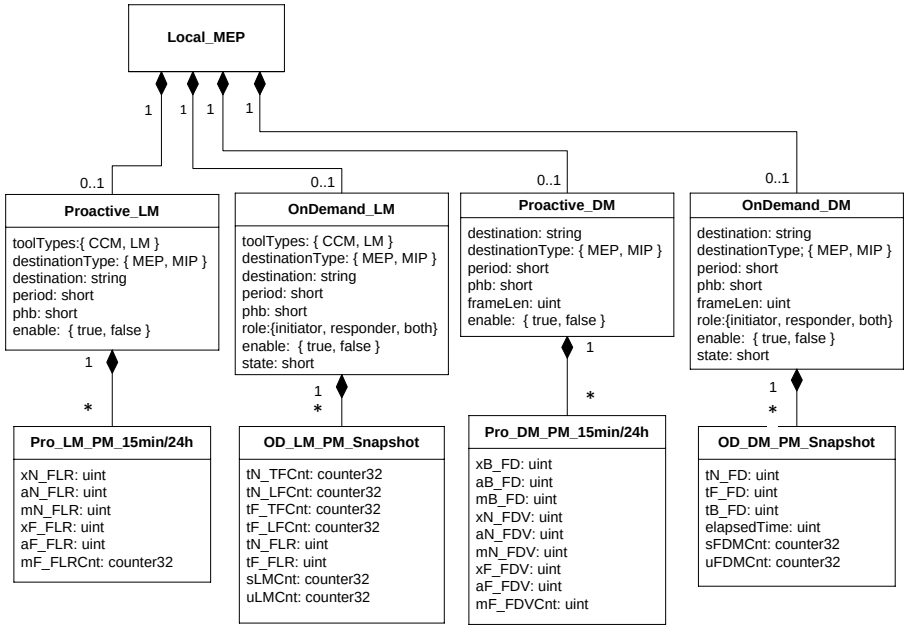

Figure 24 MPLS-TP OAM Performance Monitoring

6 Protection

This section describes the SPTN logical switch objects that support Protection.

6.1 Concepts

6.1.1 **Overall Requirements**

The SPTN switch supports PW, LSP, and section protection within the PTN network. It also supports access link (LAG) and dual-home protection. Dual-home protection combines PTN network protection and access link protection components.

6.1.2 **Protection Switching Model**

The SPTN switch supports APS mechanisms defined in ITU-T G.8031/Y.1342 [25], G.8131/Y.1382 [26], and ITU-T G.8132/Y.1344 [24].

Protection switching at the packet flow level uses OpenFlow Fast Failover group types to switchover from working to protection paths. For 1:1 protection, multiple levels of protection are supported using multiple failover groups interspersed between indirect groups that push labels.

Switchover is controlled by a local Protection Switching process that runs the APS protocol and maintains the APS state machines. This process is fed by signal fail (SF) and signal degrade (SD) indications from the OAM processing function and external commands.

Switchover is implemented using OpenFlow logical ports to signal protection switchover state to the switch pipeline. The OpenFlow Fast Failover group is defined to have an ordered set of action buckets such that the first “live” bucket is selected. The OpenFlow 1.3.4 specification defines liveness monitoring to determine which bucket to use for forwarding. Fast Failover

groups can be configured with watch_port and watch_group parameters, only one of which is used to determine bucket liveness. Liveness monitoring works as follows:

- A port is considered live if it has the OFPPS_LIVE flag set in its port state. Port liveness may be managed by code outside of the OpenFlow portion of a switch or defined outside of the OpenFlow specification. The port must not be considered live (and the OFPPS_LIVE flag must be unset) if one of the port liveness mechanisms enabled on the switch consider the port not live, or if the port config bit OFPPC_PORT_DOWN indicates the port is down, or if the port state bit OFPPS_LINK_DOWN indicates the link is down.
- A bucket is considered live if either watch_port is not OFPP_ANY and the port watched is live, or if watch_group is not OFPG_ANY and the group watched is live.
- A group is considered live if a least one of its buckets is live.

The SPTN switch uses OAM Protection Liveness Logical Ports solely for the purpose of controlling the liveness property for Fast Failover group entry buckets. Local protection processing can cause one or more Fast Failover groups to switch buckets by changing the administrative state of an OAM Protection Liveness Logical Port to down.

The protection switching process and its linkage to the pipeline for switchover is illustrated below.

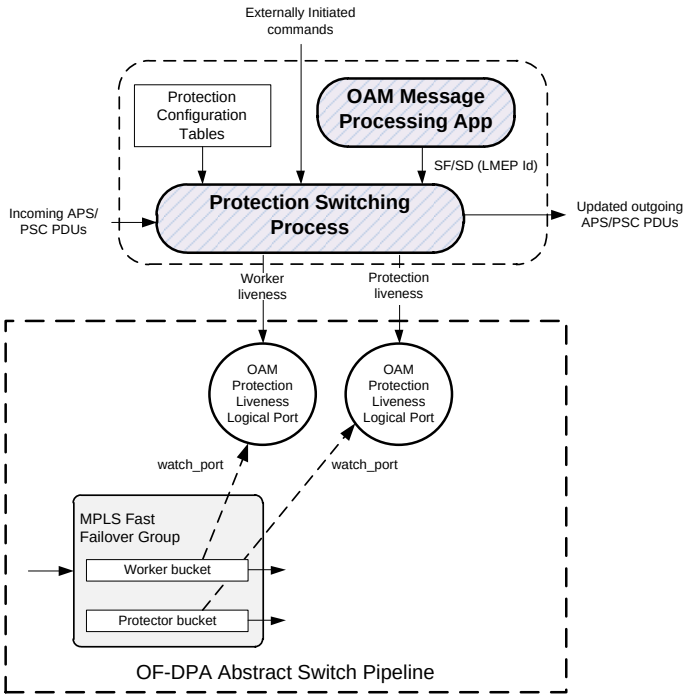

Figure 25 Protection Switching Model

OAM Protection Liveness Logical ports are predefined and do not need to be configured before use.

By convention, worker buckets have index (1) and protection buckets have index (0), even though worker buckets must appear first in configuration messages.

6.2 Packet Flows

The SPTN switch supports up to two levels of 1:1 linear protection using Fast Failover group type entries.

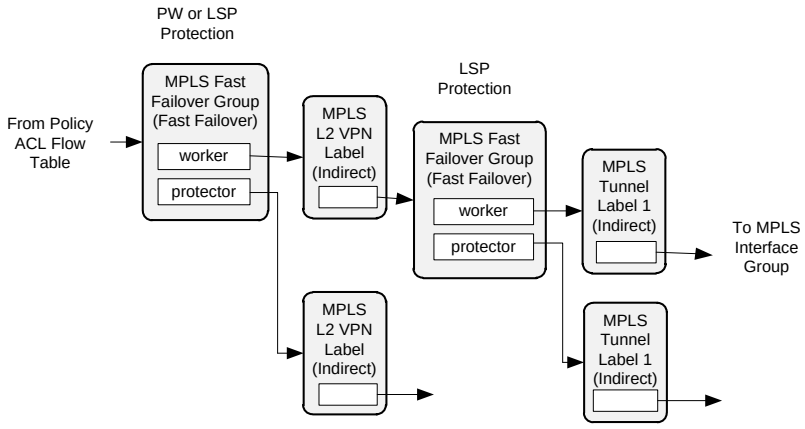

Figure 26 Fast Failover Groups

The following group table entry types are used for protection:

- **MPLS Fast Failover**. This OpenFlow Fast Failover group entry type contains two buckets whose liveness is controlled by OAM Protection Liveness Logical port liveness state. The buckets are ordered, first the "working" bucket and then the "protector" bucket. Each must have an OAM Protection Liveness Logical Port assigned as its watch_port parameter.

6.3 OpenFlow Extensions

The following pipeline match field is used in conjunction with Protection switchover.

- **Protection_Index**. An optional field associated with an MPLS label that can be set to indicate whether it is for a working (1) or protection (0) path.

6.4 Configuration Extensions

6.4.1 Linear Protection

For linear protection the LSP and PW are protected objects.

As before the Protection Group is an OF-CONFIG resource instance.

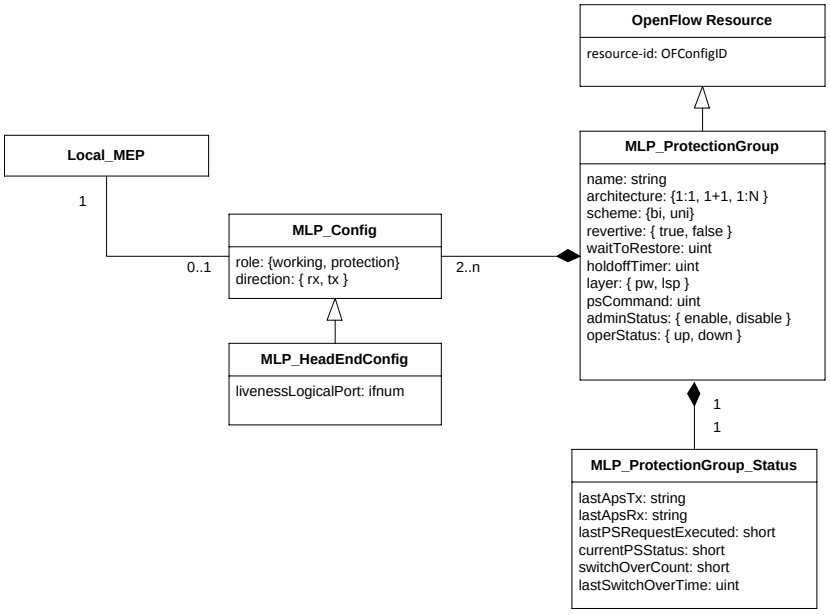

Figure 27 Linear Protection

7 Statistics

7.1 Concepts

Detailed statistics are required for performance monitoring and for providing feedback to end users.

Statistics counters are required on the following:

- Physical ports
- Services/EVCs
- MPLS Layers (PW/LSP/Section)

This section specifies the statistics counter requirements.

7.1.1 **Port Statistics**

The OpenFlow specification lists the per-port counters in Table 2, many of which are indicated as optional. This specification considers all as required.

Table 2 OpenFlow Port Statistics

Name	**Bits**	**Description**
Received Packets	64	Total packets received
Transmitted Packets	64	Total packets transmitted
Received Bytes	64	Total bytes received

Name	Bits	Description
Transmitted Bytes	64	Total bytes transmitted
Receive Drops	64	Received packets dropped for any reason
Transmit Drops	64	Transmitted packets dropped for any reason
Receive Errors	64	Received packet errors
Transmit Errors	64	Transmit packets errors
Receive Frame Alignment Errors	64	Received packets with frame alignment errors
Receive Overrun Errors	64	Received packet overruns
Receive CRC Errors	64	Received packet CRC errors
Collisions	64	Collisions
Duration (sec)	32	Time in seconds since configured

7.1.2 Service Statistics

Table 3 lists the required per-service statistics. These are collected at UNI ports. They must be maintained by service and Traffic Class, and may optionally be also maintained by Color (G/Y).

Table 3 Service Statistics

Name	Bits	Description
Received Packets	64	Total packets received
Transmitted Packets	64	Total packets transmitted
Received Bytes	64	Total bytes received
Transmitted Bytes	64	Total bytes transmitted
Duration (sec)	32	Time in seconds since configured

7.1.3 MPLS Layer Statistics

Table 4 lists the required statistics to be maintained per pseudo-wire and LSP. These are collected at NNI ports. These must be maintained by layer and Color (G/Y).

Table 4 MPLS Layer Statistics

Name	Bits	Description
Received Packets	64	Total packets received
Transmitted Packets	64	Total packets transmitted
Received Bytes	64	Total bytes received
Transmitted Bytes	64	Total bytes transmitted
Duration (sec)	32	Time in seconds since configured

7.2 Packet Flows

This section specifies instrumentation points and mechanisms that meet the statistics gathering requirements.

7.2.1 **VPWS**

This section describes the instrumentation for VPWS flows.

7.2.1.1 ***Attachment Circuit (UNI->NNI) Counters***

Figure 28 shows the added statistics counters for the UNI->NNI initiation use case.

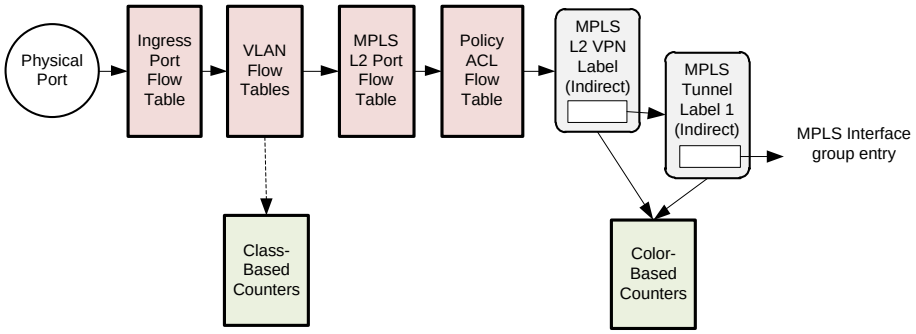

Figure 28 VPWS UNI->NNI Flow with Counter Objects

The following counter objects are introduced and highlighted in green in the diagram:

- **Class-Based Counters.** These are stateful functions that provide transmit and receive packet and byte counters for eight traffic classes. A counter block index references a set of 32 counters. Individual counters can be referenced by counter block index, direction, and Traffic Class pipeline field value.

  Class-Based counters are used to gather statistics for individual customer service flows.

  For initiation flows they are updated by the VLAN Flow Table using the Class_Based_Count action and specifying a counter block index. The Traffic Class is supplied by the switch corresponding to the packet Traffic Class pipeline field value at the point where the counter is incremented. Note that the action must be in the Action Set in order to update the counter for the final Traffic Class value after traffic classification and policing.

- **Color-Based Counters**. These are stateful functions that provide transmit and receive packet and byte counters by packet Color. Yellow and Green counters are supported. A counter block index references a set of eight counters.

  For initiation flows the Color-Based transmit counters are updated by Color_Based_Count actions in MPLS VPN and MPLS Tunnel group entry bucket action sets for the relevant MPLS protocol layer. The Controller assigns a counter block index to each pseudo-wire or LSP.

7.2.1.2 ***Network Interface (NNI->UNI) Counters***

Figure 29 shows the added statistics counters for the NNI->UNI termination use case.

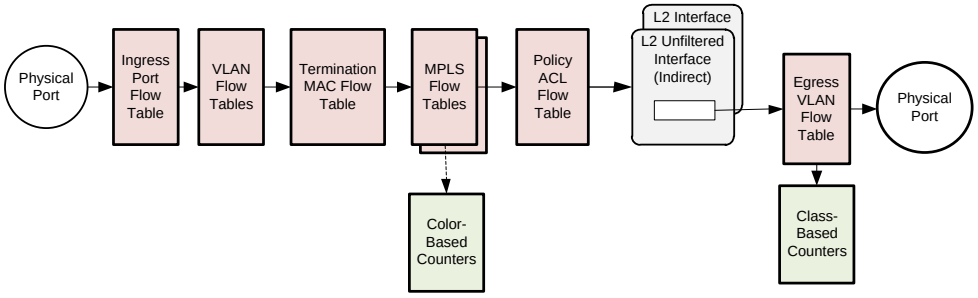

Figure 29 VPWS NNI->UNI Flow with Counter Objects

The following counter objects are introduced and highlighted in green in the diagram:

- **Class-Based Counters.** Stateful functions that provide transmit and receive packet and byte counters by traffic class for individual customer services.

  For termination flows Class-Based counters are updated by Class_Based_Count actions invoked from the Egress VLAN Flow Table in either the Action Set or Action List.

- **Color-Based Counters**. Stateful functions that provide transmit and receive packet and byte counters by packet Color for MPLS protocol layers (PW, LSP, Section).

  For termination flows Color-Based receive counters are updated by Color_Based_Count actions in MPLS Flow Table entries that process different MPLS protocol layers. The Controller assigns a counter block index to each pseudo-wire or LSP.

7.2.2 **LSR**

Figure 30 shows the added statistics counters for the LSR use case.

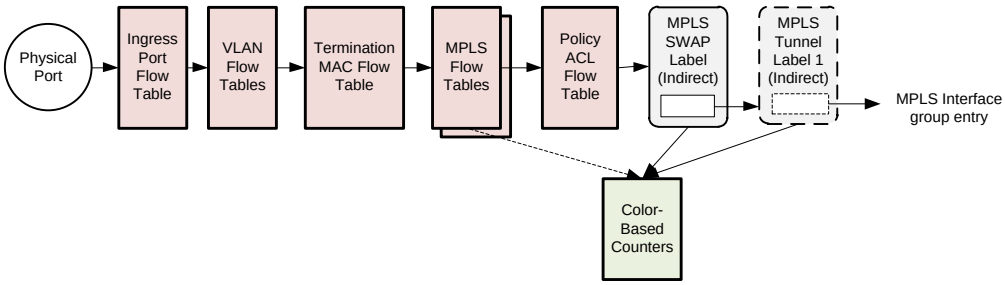

Figure 30 LSR Flow with Counter Objects

The following counter objects are used in Figure 30, highlighted in green:

- **Color-Based Counters**. Stateful functions that provide transmit and receive packet and byte counters by packet Color for MPLS protocol layers (PW, LSP, Ring). Packet color is determined based on the EXP field in the MPLS shim header set using the MPLS Set QoS Action table as described in the QoS section.

For ingress packets MPLS Flow Table rules the Action Set can include Color_Based_Count actions using the counter block index assigned by the Controller for the protocol layer.

For forwarded packets, MPLS VPN and MPLS Tunnel group entry bucket Action Sets include Color_Based_Count actions using the counter block index assigned by the Controller for the protocol layer.

7.3 OpenFlow Extensions for Statistics

7.3.1 Match Fields

No additional header match fields are needed for statistics.

7.3.2 Pipeline Match Fields

No additional pipeline match fields are needed for statistics.

7.3.3 Actions

The following actions are introduced.

- **Class_Based_Count**. Update the counter indexed by the referenced counter block index and the packet Traffic Class at the time the action is executed. The switch provides the Traffic Class argument.
- **Color_Based_Count**. Update the counter indexed by the referenced counter block index and the packet Color at the time the action is executed. The switch provides the Color argument.

These require Experimenter definitions.

7.3.4 Objects

The following objects are introduced in conjunction with the actions described in the previous section.

- **Class-Based Counter Table.** Counter objects that provide a block of 32 counters consisting of transmit and receive packet and byte counters for eight values of Traffic Class. Each block is referenced by a counter block index.
- **Color-Based Counter Table**. Counter objects that provide a block of eight counters consisting of transmit and receive packet and byte counters for two Color values. Each block is referenced by a counter block index.

These require Experimenter protocol definitions as well as new Experimenter message types for modifying and querying entries.

7.4 Configuration Extensions

No configuration extensions are required to support statistics.

8 Notifications

Notification and alarm augmentations to OF-CONFIG 2.0 are provided in the accompanying YANG file named “of-config-1.2-sptn-alarm.yang”.

9 Experimenter Encodings

This section describes experimenter protocol extensions to OpenFlow.

Experimenter headers require an Experimenter ID. The following value is assigned from the ONF managed Experimenter ID space.

```
enum sptn_experimenter_id {
    SPTN_EXPERIMENTER_ID =        FF-00-00-0A
};
```

Experimenter type definitions are as follows:

```
enum sptn_experimenter_type {
    SPTNEXP_OAM_DATA_PLANE_COUNTER_MOD_MSG =          2,
    SPTNEXP_OAM_DATA_PLANE_COUNTER_MULTIPART =        3,
    SPTNEXP_DROP_STATUS_MOD_MSG =                     4,
    SPTNEXP_DROP_STATUS_MULTIPART =                   5,
    SPTNEXP_MPLS_VPN_LABEL_REMARK_MOD_MSG =           6,
    SPTNEXP_MPLS_VPN_LABEL_REMARK_MULTIPART =         7,
    SPTNEXP_MPLS_TUNNEL_LABEL_REMARK_MOD_MSG =        8,
    SPTNEXP_MPLS_TUNNEL_LABEL_REMARK_MULTIPART =      9,
    SPTNEXP_CLASS_BASED_CTR_MOD_MSG =                 12,
    SPTNEXP_CLASS_BASED_CTR_MULTIPART =               13,
    SPTNEXP_COLOR_BASED_CTR_MOD_MSG =                 14,
    SPTNEXP_COLOR_BASED_CTR_MULTIPART =               15,
    SPTNEXP_QUEUE_MOD_MSG =                           17,
    SPTNEXP_QUEUE_DESC_PROP_WEIGHT =                  18,
    SPTNEXP_QUEUE_DESC_PROP_CONGESTION =              19
};

enum sptn_msg_mod_command {
    SPTN_MSG_MOD_ADD =                                0,
    SPTN_MSG_MOD_MODIFY =                             1,
    SPTN_MSG_MOD_DELETE =                             2,
};

enum sptn_color_values {
    SPTN_COLOR_GREEN =                  0,
    SPTN_COLOR_YELLOW =                 1,
    SPTN_COLOR_RED =                    2,
    SPTN_COLOR_ALL =                    0xFF,
}

enum sptn_traffic_class_values {
    SPTN_TRAFFIC_CLASS_BE =             0,
    SPTN_TRAFFIC_CLASS_AF1 =            1,
    SPTN_TRAFFIC_CLASS_AF2 =            2,
    SPTN_TRAFFIC_CLASS_AF3 =            3,
    SPTN_TRAFFIC_CLASS_AF4 =            4,
    SPTN_TRAFFIC_CLASS_EF =             5,
    SPTN_TRAFFIC_CLASS_CS6 =            6,
    SPTN_TRAFFIC_CLASS_CS7 =            7,
    SPTN_TRAFFIC_CLASS_ALL =            0xFF,
}

enum sptn_all_any_values {
    SPTN_LMEP_ALL =                     0xFFFFFFFF,
    SPTN_INDEX_ALL =                    0xFFFFFFFF,
    SPTN_DROP_STATUS_ANY =              0xFF,
    SPTN_MPLS_TC_ALL =                  0xFF,
```



```
    SPTN_VLAN_PCP_ALL =                 0xFF,
    SPTN_VLAN_DEI_ALL =                 0xFF,
};
```

9.1 OAM Data Plane Counter Table

The OAM Data Plane Counter Table is defined to maintain the LM counters needed for OAM processing. Entries in this table implement a globally accessible LM counter resource. They can be updated using actions (OAM_LM_Tx_Count, OAM_LM_Rx_Count) from flow tables and group entries and read by the local OAM processor. Actions are defined to update LM counters.

The OAM Data Plane Counter Table is indexed by openFlowMpId and Traffic Class. Each entry can hold a packet counter for transmit and receive directions. Furthermore each entry maintains a reference count.

An experimenter message type is used to support modifications to the OAM Data Plane Counter Table. This is only used as a Controller to Switch message. An entry in the table should be created for each configured openFlowMpId.

```
struct sptn_oam_data_plane_ctr_mod_msg {
    struct ofp_experimenter_header header; /*16 bytes, SPTN_EXPERIMENTER_ID */
                          /* exp_type = SPTNEXP_OAM_DATA_PLANE_COUNTER_MOD_MSG */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        command;            /* one of SPTN_MSG_MOD_* */
    uint32_t        lmep_id;            /* index into table entry */
    uint64_t        transmit_packets;   /* must be zero, can clear only */
    uint64_t        receive_packets;    /* must be zero, can clear only */
    uint8_t         traffic_class;      /* SPTN_TRAFFIC_CLASS_* */
    uint8_t         pad[7];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_oam_data_plane_ctr_mod_msg) == 48);
```

An experimenter multipart message type is used to read status of the OAM Data Plane Counter Table. It is used for both request and reply messages.

```
struct sptn_oam_data_plane_ctr_multipart_request {
    struct ofp_multipart_request  header; /*16 bytes, OFPMP_EXPERIMENTER */

    /* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
    struct ofp_experimenter_multipart_header exp_hdr;/* 8 bytes */
                          /* exp_type = SPTNEXP_OAM_DATA_PLANE_COUNTER_MULTIPART*/

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        lmep_id;            /* SPTN_LMEP_ALL to get all lmep id */
    uint8_t         traffic_class;      /* SPTN_TRAFFIC_CLASS_ALL to get all */
    uint8_t         pad_1[3];           /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_OAM_DATA_PLANE_CTR_multipart_request) == 32);


struct sptn_oam_data_plane_ctr_multipart_reply {
....struct ofp_multipart_reply   header; /* 16 bytes, OFPMP_EXPERIMENTER */

    /* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
    struct ofp_experimenter_multipart_header exp_hdr;/* 8 bytes */
```



```
                        /* exp_type = SPTNEXP_OAM_DATA_PLANE_COUNTER_MULTIPART*/

    /* Experimenter-defined arbitrary additional data. */
    uint64_t        transmit_packets;   /* transmitted packets */
    uint64_t        receive_packets;    /* received packets */
    uint64_t        reference_count;    /* reference count */
    uint32_t        lmep_id;            /* lmep id of entry */
    uint8_t         traffic_class;      /* traffic class of entry */
    uint8_t         pad_1[3];           /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_OAM_DATA_PLANE_CTR_multipart_reply) == 56);
```

9.2 Drop Status Action Table

The Drop Status Action Table is referenced by a Check Drop Status action with index and type arguments. Currently only access by openFlowMpId and type zero are supported. It is used for the OAM LCK function data plane operation in order to drop OAM and data frames during a lock condition.

The Drop Status Table can be considered an auxiliary object similar to a Meter entry with “drop” band except that the drop action is controlled by the Network Protection Process using an unspecified internal interface, similar to what is done for the switchover control on the OAM Protection Liveness Logical port. The default action on a Check Drop-Status action lookup miss is “Do not drop.”

An experimenter message type is used to support modifications to the Drop Status Table. This is only used as a Controller to Switch message. An entry in the table should be created for each configured openFlowMpId. Note that the drop status can only be initialized by the Controller to zero. If drop status of a MEP configured with the corresponding openFlowMpId value changes, the local processing function changes the value for that entry in the Drop Status Table.

```
struct sptn_drop_status_mod_msg {
    struct ofp_experimenter_header header; /*16 bytes, SPTN_EXPERIMENTER_ID */
                        /* exp_type = SPTNEXP_DROP_STATUS_MOD_MSG */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        command;            /* one of SPTN_MSG_MOD_* */
    uint32_t        index;              /* index to entry in table */
    uint8_t         entry_type;         /* type of entry, only 0 valid here */
    uint8_t         drop_status;        /* 0=do not drop; 1= drop */
    uint8_t         pad[6];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_drop_status_mod_msg) == 32);
```

An experimenter multipart message type is used to read status of the Drop Status Table. It is used for both request and reply messages.

```
struct sptn_drop_status_multipart_request {
    struct ofp_multipart_request  header; /*16 bytes, OFPMP_EXPERIMENTER */

    /* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
    struct ofp_experimenter_multipart_header exp_hdr;/* 8 bytes */
                        /* exp_type = SPTNEXP_DROP_STATUS_MULTIPART */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        index;              /* SPTN_INDEX_ALL to get all entries*/
```



```
    uint8_t         type;               /* must be 1, others reserved */
    uint8_t         pad_1[3];           /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_drop_status_multipart) == 32);


struct sptn_drop_status_multipart_reply {
    struct ofp_multipart_request  header; /*16 bytes, OFPMP_EXPERIMENTER */

    /* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
    struct ofp_experimenter_multipart_header exp_hdr; /* 8 bytes */
                          /* exp_type = SPTNEXP_DROP_STATUS_MULTIPART */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        index;              /* index */
    uint8_t         type;               /* always 1 */
    uint8_t         drop_status         /* drop status */
    uint8_t         pad_1[2];           /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_drop_status_multipart) == 32);
```

9.3 MPLS Label Remark Action Tables

The MPLS Label Remark action tables can be invoked from the MPLS Label Group entries using optional Set-MPLS-TC-From-Table and Set-MPLS-PCP-DEI-From-Table actions. The Set-MPLS-TC-From-Table action uses the packet Traffic Class and Color and a supplied QoS Index argument to effectively perform a Set-Field (MPLS_TC) action on the outermost label. The Set-MPLS-PCP-DEI-From-Table action uses the packet Traffic Class and Color along with a supplied QoS Index argument to effectively perform Set-Field (PCP) and Set-Field (DEI) actions on the outermost VLAN tag.

While similar to match-action tables in some respects, these cannot be flow tables since they are invoked in the context of evaluating an Action Set in a group entry to perform Set-Field actions. These tables are modified and read using new Experimenter messages.

There are two MPLS Label Remark action tables:

- MPLS VPN Label Remark Table
- MPLS Tunnel Label Remark Table

An experimenter message type is used to support modifications to the MPLS VPN Label Remark Table. This is only used as a Controller to Switch message.

```
struct sptn_mpls_vpn_label_remark_action_mod_msg {
    struct ofp_experimenter_header header; /*16 bytes, SPTN_EXPERIMENTER_ID */
                          /* exp_type = SPTNEXP_MPLS_VPN_LABEL_REMARK_MOD_MSG */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        command;            /* one of SPTN_MSG_MOD_* */
    uint32_t        index;              /* index */
    uint8_t         traffic_class;      /* traffic class */
    uint8_t         color;              /* color */
    uint8_t         mpls_tc;            /* MPLS TC value to set */
    uint8_t         vlan_pcp;           /* outer vlan PCP to set */
    uint8_t         vlan_dei;           /* outer vlan DEI to set */
```



```
    uint8_t         pad[3];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_mpls_vpn_label_remark_action_mod_msg) == 32);
```

An experimenter multipart message type is used to read status of the MPLS VPN Label Remark Table. It is used for both request and reply messages.

```
struct sptn_mpls_vpn_label_remark_action_multipart_request {
    struct ofp_multipart_request  header; /*16 bytes, OFPMP_EXPERIMENTER */

    /* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
    struct ofp_experimenter_multipart_header exp_hdr; /* 8 bytes */
                          /* exp_type = SPTNEXP_MPLS_VPN_LABEL_REMARK_MULTIPART */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        index;              /* SPTN_INDEX_ALL to get all */
    uint8_t         traffic_class;      /* SPTN_TRAFFIC_CLASS_ALL to get all */
    uint8_t         color               /* SPTN_MPLS_COLOR_ALL to get all*/
    uint8_t         pad[2]              /* align message on 64-byte boundary */
};
OFP_ASSERT(sizeof(struct sptn_mple_vpn_label_remark_action_multipart_request)== 32);


struct sptn_mpls_vpn_label_remark_action_multipart_reply {
    struct ofp_multipart_reply   header; /*16 bytes, OFPMP_EXPERIMENTER */

    /* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
    struct ofp_experimenter_multipart_header exp_hdr; /* 8 bytes */
                          /* exp_type = SPTNEXP_MPLS_VPN_LABEL_REMARK_MULTIPART */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        index;              /* entry index */
    uint8_t         traffic_class;      /* entry traffic class */
    uint8_t         color;              /* entry color */
    uint8_t         mpls_tc             /* entry mpls_tc value */
    uint8_t         vlan_pcp            /* entry vlan_pcp value */
    uint8_t         vlan_dei            /* engry vlan_dei value */
    uint8_t         pad[7];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_mple_vpn_label_remark_action_multipart_reply)==40);
```

An experimenter message type is used to support modifications to the MPLS Tunnel Label Remark Table. This is only used as a Controller to Switch message.

```
struct sptn_mpls_tunnel_label_remark_mod_msg {
    struct ofp_experimenter_header header; /*16 bytes, SPTN_EXPERIMENTER_ID */
                          /* exp_type = SPTNEXP_MPLS_TUNNEL_LABEL_REMARK_MOD_MSG */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        command;            /* one of SPTN_MSG_MOD_* */
    uint32_t        index;              /* index */
    uint8_t         traffic_class;      /* traffic class */
    uint8_t         color;              /* color */
    uint8_t         mpls_tc;            /* MPLS TC value to set */
    uint8_t         vlan_pcp;           /* outer vlan PCP to set */
    uint8_t         vlan_dei;           /* outer vlan DEI to set */
    uint8_t         pad[3];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_mpls_tunnel_label_remark_action_mod_msg)==32);
```


An experimenter multipart message type is used to read status of the MPLS Tunnel Label Remark Table. It is used for both request and reply messages.

```
struct sptn_mpls_tunnel_label_remark_action_multipart_request {
    struct ofp_multipart_request  header; /*16 bytes, OFPMP_EXPERIMENTER */

    /* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
    struct ofp_experimenter_multipart_header exp_hdr; /* 8 bytes */
                          /* exp_type = SPTNEXP_MPLS_TUNNEL_LABEL_REMARK_MULTIPART */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        index;              /* SPTN_INDEX_ALL to get all */
    uint8_t         traffic_class;      /* SPTN_TRAFFIC_CLASS_ALL to get all */
    uint8_t         mpls_tc             /* SPTN_MPLS_TC_ALL to get all*/
    uint8_t         vlan_pcp            /* SPTN_VLAN_PCP_ALL to get all */
    uint8_t         vlan_dei            /* SPTN_VLAN_DEI_ALL to get all*/
};
OFP_ASSERT(sizeof(struct sptn_mple_tunnel_label_remark_action_multipart_request)==32);


struct sptn_mpls_tunnel_label_remark_action_multipart_reply {
    struct ofp_multipart_reply   header; /*16 bytes, OFPMP_EXPERIMENTER */

    /* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
    struct ofp_experimenter_multipart_header exp_hdr; /* 8 bytes */
                          /* exp_type = SPTNEXP_MPLS_TUNNEL_LABEL_REMARK_MULTIPART */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        index;              /* entry index */
    uint8_t         traffic_class;      /* entry traffic class */
    uint8_t         mpls_tc             /* entry mpls_tc value */
    uint8_t         vlan_pcp            /* entry vlan_pcp value */
    uint8_t         vlan_dei            /* engry vlan_dei value */
};
OFP_ASSERT(sizeof(struct sptn_mple_tunnel_label_remark_action_multipart_reply)==32);
```

9.4 Class Based Counter Table

The Class Based Counter Table is defined to maintain packet and byte statistics counters by Traffic Class. Entries in this table implement a globally accessible counter resource. They can be updated using the Class_Based_Count action from flow tables and group entries and read by the Controller. The Class_Based_Count action updates counters based on a counter block index and Traffic Class.

The Class Based Counter Table is indexed by counter block index and Traffic Class. Each entry can hold a packet counter for transmit and receive directions. Furthermore each entry maintains a reference count.

An experimenter message type is used to support modifications to the Class Based Counter Table. This is only used as a Controller to Switch message. An entry in the table should be created for each service flow.

```
struct sptn_class_based_ctr_mod_msg {
    struct ofp_experimenter_header header; /*16 bytes, SPTN_EXPERIMENTER_ID */
                          /* exp_type = SPTNEXP_CLASS_BASED_CTR_MOD_MSG */

    /* Experimenter-defined arbitrary additional data. */
```



```
    uint32_t        command;            /* one of SPTN_MSG_MOD_* */
    uint32_t        block_index;        /* index into table entry */
    uint64_t        packets;            /* clear only */
    uint64_t        bytes;              /* clear only */
    uint8_t         traffic_class;      /* sub-index into table entry */
    uint8_t         pad[7];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_class_based_ctr_mod_msg) == 48);
```

An experimenter multipart message type is used to read status of the Class Based Counter Table. It is used for both request and reply messages.

```
struct sptn_class_based_ctr_multipart_request {
    struct ofp_multipart_request header; /*16 bytes, OFPMP_EXPERIMENTER */

    /* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
    struct ofp_experimenter_multipart_header exp_hdr; /* exp_type =
                          SPTNEXP_OAM_CLASS_BASED_CTR_MULTIPART, 8 bytes */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        block_index;        /* can be SPTN_INDEX_ALL */
    uint8_t         traffic_class;      /* SPTN_TRAFFIC_CLASS_ALL to get all */
    uint8_t         pad[3];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_class_based_ctr_multipart_request) == 32);


struct sptn_class_based_ctr_multipart_reply {
    struct ofp_multipart_request header; /*16 bytes, OFPMP_EXPERIMENTER */

    /* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
    struct ofp_experimenter_multipart_header exp_hdr; /* exp_type =
                          SPTNEXP_OAM_CLASS_BASED_CTR_MULTIPART, 8 bytes */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        block_index;        /* entry index */
    uint8_t         pad_1[4];           /* pad to 64 bit boundary */
    uint64_t        packets;            /* packet count */
    uint64_t        bytes;              /* byte count */
    uint64_t        reference_count;    /* reference count */
    uint8_t         traffic_class;      /* SPTN_TRAFFIC_CLASS_* */
    uint8_t         pad_2[7];           /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_class_based_ctr_multipart_reply) == 64);
```

9.5 Color Based Counter Table

The Color Based Counter Table is defined to maintain packet and byte statistics counters for two Colors, Yellow and Green. Entries in this table implement a globally accessible counter resource. They can be updated using Color_Based_Count actions from flow tables and group entries and read by the Controller. The Color_Based_Count action updates counters based on a counter block index and packet Color.

The Color Based Counter Table is indexed by counter block index and Color. Each entry can hold a packet counter for transmit and receive directions. Furthermore each entry maintains a reference count.

An experimenter message type is used to support modifications to the Color Based Counter Table. This is only used as a Controller to Switch message. An entry in the table should be created for each service flow.

```
struct sptn_color_based_ctr_mod_msg {
    struct ofp_experimenter_header header; /*16 bytes, SPTN_EXPERIMENTER_ID */
                           /* exp_type = SPTNEXP_COLOR_BASED_CTR_MOD_MSG */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        command;            /* one of SPTN_MSG_MOD_* */
    uint32_t        block_index;        /* index into table entry */
    uint64_t        transmit_packets;   /* clear only */
    uint64_t        receive_packets;    /* clear only */
    uint8_t         color;              /* sub-index into table entry */
    uint8_t         pad[7];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_color_based_ctr_mod_msg) == 48);
```

An experimenter multipart message type is used to read status of the Color Based Counter Table. It is used for both request and reply messages.

```
struct sptn_color_based_ctr_multipart_request {
    struct ofp_multipart_request  header; /*16 bytes, OFPMP_EXPERIMENTER */

    /* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
    struct ofp_experimenter_multipart_header exp_hdr; /* exp_type =
                           SPTNEXP_OAM_COLOR_BASED_CTR_MULTIPART, 8 bytes */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        block_index;        /* can be SPTN_INDEX_ALL */
    uint8_t         color;              /* SPTN_COLOR_ALL to get all */
    uint8_t         pad[3];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_class_based_ctr_multipart_request) == 32);


struct sptn_color_based_ctr_multipart_reply {
    struct ofp_multipart_request  header; /*16 bytes, OFPMP_EXPERIMENTER */

    /* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
    struct ofp_experimenter_multipart_header exp_hdr; /* exp_type =
                           SPTNEXP_OAM_COLOR_BASED_CTR_MULTIPART, 8 bytes */

    /* Experimenter-defined arbitrary additional data. */
    uint32_t        block_index;        /* entry index */
    uint8_t         pad_1[4];           /* pad to 64 bit boundary */
    uint64_t        packets;            /* packet count */
    uint64_t        bytes;              /* byte count */
    uint64_t        reference_count;    /* reference count */
    uint8_t         color;              /* SPTN_COLOR_* */
    uint8_t         pad_2[7];           /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_class_based_ctr_multipart_reply) == 64);
```

9.6 Queue Modification and Properties

An experimenter message type is defined for a Queue Modification message. This is used so that queue properties can be dynamically updated using OpenFlow. Experimenter queue description

properties are also defined for the relationship between queues, ports and scheduler nodes in the QoS hierarchy.

```
struct sptn_queue_mod_msg {
    struct ofp_experimenter_header header; /*16 bytes, SPTN_EXPERIMENTER_ID */
                           /* exp_type = SPTN_EXP_QUEUE_MOD_MSG */

/* Experimenter-defined arbitrary additional data. */
    uint32_t        command;            /* One of SPTN_MSG_MOD_* */
    uint8_t         pad[4];             /* 64-bit boundary */
    struct ofp_packet_queue  queue;     /* Queue without properties – 16 bytes */
};
OFP_ASSERT (size(struct sptn_queue_mod_msg) == 40);
```

The queue description specifies the queue number and port.

Queue properties for max rate and min rate are defined in the OpenFlow 1.3.4 specification and are applicable here[11]. Experimenter property types are defined for scheduling weight (for DWRR and WRR) and for congestion. Congestion curve parameters must be supplied for each packet color if the congestion mode is WRED.

The following congestion property specifies the congestion mode and parameters for the queue. If this property is not supplied the default is `SPTN_QDC_TAIL_DROP`.

```
enum sptn_queue_desc_congestion {
    SPTN_QDC_TAIL_DROP =         1, /* Tail drop */
    SPTN_QDC_WRED =              2  /* Weighted Random Early Discard */
};

/* OFPQDPT_EXPERIMENTER queue property for congestion. */
struct sptn_queue_desc_prop_congestion {
    struct ofp_queue_prop_header prop_header; /* OFPQT_EXPERIMENTER */
    uint32_t experimenter;       /* Experimenter ID which takes the same
                                 form as in struct ofp_experimenter_header. */
    uint8_t         pad[4];      /* Pad to 64 bit alignment */

/* Experimenter defined data */
    uint16_t mode;               /* One of SPTN_QDC_* */
    uint8_t  pad1[2];            /* Pad to 32-bit alignment */
    uint32_t weight;             /* Scheduling weight for WRR and WDRR */
    uint32_t green_drop_start;   /* Green packet WRED curve drop start */
    uint32_t green_drop_end;     /* Green packet WRED curve drop end*/
    uint32_t green_drop_slope;   /* Green packet WRED angle as a ratio */
    uint32_t yellow_drop_start;  /* Yellow packet WRED curve drop start */
    uint32_t yellow_drop_end;    /* Yellow packet WRED curve drop end*/
    uint32_t yellow_drop_slope;  /* Yellow packet WRED angle as a ratio */
    uint32_t red_drop_start;     /* Red packet WRED curve drop start */
    uint32_t red_drop_end;       /* Red packet WRED curve drop end*/
    uint32_t red_drop_slope;     /* Red packet WRED angle as a ratio */
    uint8_t  pad2[4];            /* Pad to 64 bit alignment */
};
OFP_ASSERT (size(struct sptn_queue_desc_prop_congestion) == 64);

struct sptn_queue_desc_prop_congestion_mod_msg {
    struct ofp_experimenter_header header; /*16 bytes, SPTN_EXPERIMENTER_ID */
                           /* exp_type = SPTN_EXP_QUEUE_DESC_PROP_CONGESTION */
```

[11] However it would be better to express rates as absolute values in kb/sec rather than relative as in OpenFlow 1.3.4.


```
    /* Experimenter-defined arbitrary additional data. */
    uint32_t  command;                  /* One of SPTN_MSG_MOD_* */
    uint8_t   pad[4];                   /* Pad to 64 bytes */
    struct ofp_packet_queue queue;      /* Queue definition, 16 bytes*/
    struct sptn_queue_desc_prop_congestion prop_congestion; /* Congestion */
    uint32_t scheduling_priority;       /* Scheduling priority */
};
OFP_ASSERT (size(struct sptn_queue_desc_prop_congestion_mod_msg) == 98);
```

9.7 Color Set Meter Band

The experimenter definitions to support color set meter bands is given below. This is the only meter band type supported and replaces the corresponding fields in Meter modification messages. Both color set meter bands must be the same mode.

```
/* SPTN Experimenter Meter Band types */
enum sptn_meter_band_exp_type {
       SPTN_OFPMBT_COLOR_SET =                1
 };

/* SPTN Experimenter Color Set Meter Band Modes */
enum sptn_color_set_band_mode {
       SPTN_COLOR_SET_BAND_MODE_TRTCM =        1,
       SPTN_COLOR_SET_BAND_MODE_SRTCM =        2,
       SPTN_COLOR_SET_BAND_MODE_RFC4115 =      3
};

enum sptn_color_aware {
       SPTN_COLOR_AWARE_BLIND =                0,
       SPTN_COLOR_AWARE_AWARE =                1
};

/* OFPMT_EXPERIMENTER band for Color Set */

struct ofp_meter_band_experimenter_color_set {
    struct ofp_meter_band_experimenter band; /* 16 bytes, SPTN_EXPERIMENTER_ID */
    uint16_t        exp_type;           /* OFPMBT_COLOR_SET (1) */
    uint8_t         mode;               /* One of SPTN_COLOR_SET_BAND_MODE_* */
    uint8_t         color-aware;        /* One of SPTN_COLOR_AWARE_*  */
    uint8_t         color;              /* New color, one of SPTN_COLOR_* */
    unit8_t         pad[3];             /* Align on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct ofp_meter_band_experimenter_color_set) == 24);
```

9.8 Actions

The action types are programmed using the following assignments:

```
/* SPTN Experimenter Acton types */
enum sptn_action_exp_type {
    SPTN_ACT_PUSH_L2_HEADER =                          1,
    SPTN_ACT_POP_L2_HEADER =                           2,
    SPTN_ACT_PUSH_CW =                                 3,
    SPTN_ACT_POP_CW_OR_ACH =                           4,
    SPTN_ACT_COPY_TC_IN =                              5,
    SPTN_ACT_COPY_TC_OUT   =                           6,
    SPTN_ACT_SET_MPLS_TC_FROM_VPN_TABLE =              7,
```



```
    SPTN_ACT_OAM_LM_TX_COUNT =                          10,
    SPTN_ACT_OAM_LM_RX_COUNT =                          11,
    SPTN_ACT_OAM_SET_COUNTER_FIELDS =                   12,
    SPTN_ACT_DEC_TTL_MTU =                              13,
    SPTN_ACT_CHECK_DROP_STATUS    =                     14,
    SPTN_ACT_SET_MPLS_PCP_DEI_FROM_VPN_TABLE =          16,
    SPTN_ACT_SET_MPLS_TC_FROM_TUNNEL_TABLE =            17,
    SPTN_ACT_SET_MPLS_PCP_DEI_FROM_TUNNEL_TABLE =       18,
    SPTN_ACT_CLASS_BASED_COUNT =                        19,
    SPTN_ACT_COLOR_BASED_COUNT =                        20
};
```

Experimenter encodings are required for new actions. These follow the models in the OpenFlow specification and also include the experimenter header (struct_ofp_action_exprimenter_header).

Actions that have no arguments just consist of an experimenter action header and a type code value from `sptn_action_exp_type`. These are: Push L2 header; Pop L2 header; Push CW; Pop CW or ACH; Copy TC In; and Copy TC out.

```
struct sptn_action_experimenter  {
    struct ofp_action_experimenter_header header; /* 8 bytes, SPTN_EXPERIMENTER_ID */
    uint16_t        exp_type;           /* One of above action types*/
    uint8_t         pad[6];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_action_experimenter) == 16);
```

The Set MPLS TC From VPN Table sets the MPLS TC field in the PW or SWAP MPLS shim header from the value in the MPLS VPN Label Remark Table. Similarly, the Set MPLS PCP DEI From VPN Table action sets VLAN priority and drop indication header fields from a value in the MPLS VPN Label Remark Table.

```
struct sptn_action_set_mpls_tc_from_vpn_table  {
    struct ofp_action_experimenter_header header; /* 8 bytes, SPTN_EXPERIMENTER_ID */
    uint16_t        exp_type;           /* SPTN_ACT_SET_MPLS_TC_FROM_VPN_TABLE */
    uint16_t        qos_index;          /* qos index argument */
    uint8_t         traffic_class;      /* traffic class, supplied by switch */
    uint8_t         color;              /* color, supplied by switch */
    uint8_t         pad[2];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_action_set_mpls_tc_from_vpn_table) == 16);

struct sptn_action_set_mpls_pcpdei_from_vpn_table  {
    struct ofp_action_experimenter_header header; /* 8 bytes, SPTN_EXPERIMENTER_ID */
    uint16_t        exp_type;           /* SPTN_ACT_SET_MPLS_PCP_DEI_FROM_VPN_TABLE */
    uint16_t        qos_index;          /* qos index argument */
    uint8_t         traffic_class;      /* traffic class, supplied by switch */
    uint8_t         color;              /* color, supplied by switch */
    uint8_t         pad[2];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_action_set_mpls_pcpdei_from_vpn_table) == 16);
```

Similarly, Set MPLS TC From Tunnel Table sets the MPLS TC field in the MPLS LSP shim header from the value in the MPLS Tunnel Label Re-mark Table. Similarly, the Set MPLS PCP DEI From Tunnel Table action sets VLAN priority and drop indication header fields from a value in the MPLS Tunnel Label Remark Table.

```
struct sptn_action_set_mpls_tc_from_tunnel_table  {
    struct ofp_action_experimenter_header header; /* 8 bytes, SPTN_EXPERIMENTER_ID */
    uint16_t        exp_type;           /* SPTN_ACT_SET_MPLS_TC_FROM_TUNNEL_TABLE */
```



```
    uint16_t        qos_index;          /* qos index argument */
    uint8_t         traffic_class;      /* traffic class, supplied by switch */
    uint8_t         color;              /* color, supplied by switch */
    uint8_t         pad[2];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_action_set_mpls_tc_from_tunnel_table) == 16);

struct sptn_action_set_mpls_pcp_dei_from_tunnel_table  {
    struct ofp_action_experimenter_header header; /* 8 bytes, SPTN_EXPERIMENTER_ID */
    uint16_t        exp_type;    /* SPTN_ACT_SET_MPLS_PCP_DEI_FROM_TUNNEL_TABLE */
    uint16_t        qos_index;          /* qos index argument */
    uint8_t         traffic_class;      /* traffic class, supplied by switch */
    uint8_t         color;              /* color, supplied by switch */
    uint8_t         pad[2];             /* align message on 64-bit boundary */
};
OFP_ASSERT(sizeof(struct sptn_action_set_mpls_pcp_dei_from_tunnel_table) == 16);
```

The following action increments the transmit OAM loss measurement counter for the openFlowMpId and traffic class.

```
struct sptn_action_oam_lm_rx_count  {
    struct ofp_action_experimenter_header header; /* 8 bytes, SPTN_EXPERIMENTER_ID */
    uint16_t        exp_type;           /* SPTN_ACT_OAM_LM_RX_COUNT */
    uint8_t         traffic_class;      /* traffic class, supplied by switch */
    uint8_t         pad;                /* align message on 32-bit boundary */
    uint32_t        lmep_id;            /* lmep id argument */
};
OFP_ASSERT(sizeof(struct sptn_action_oam_lm_rx_count) == 16);
```

This action increments the receive OAM loss measurement counter for the openFlowMpId and traffic class.

```
struct sptn_action_oam_lm_tx_count  {
    struct ofp_action_experimenter_header header; /* 8 bytes, SPTN_EXPERIMENTER_ID */
    uint16_t        exp_type;           /* SPTN_ACT_OAM_LM_TX_COUNT */
    uint8_t         traffic_class;      /* traffic class, supplied by switch */
    uint8_t         pad;                /* align message on 32-bit boundary */
    uint32_t        lmep_id;            /* lmep id argument */
};
OFP_ASSERT(sizeof(struct sptn_action_oam_lm_tx_count) == 16);
```

This action accesses the OAM loss measurement counters for the openFlowMpId and traffic class in order to set the RxFCl and TxFCl pipeline fields for the OAM engine. It also sets the receive timestamp RxTime value for the OAM engine.

```
struct sptn_action_oam_set_counter_fields  {
    struct ofp_action_experimenter_header header; /* 8 bytes, SPTN_EXPERIMENTER_ID */
    uint16_t        exp_type;           /* SPTN_ACT_OAM_SET_COUNTER_FIELDS */
    uint8_t         traffic_class;      /* supplied by switch */
    uint8_t         pad;                /* align message on 32-bit boundary */
    uint32_t        lmep_id;            /* lmep id argument */
};
OFP_ASSERT(sizeof(struct sptn_action_oam_set_counter_fields) == 16);
```

The Check Drop Status action accesses the drop status table for the given index and type. If an entry is found with drop_status equal to 1 the packet is immediately dropped.

```
struct sptn_action_check_drop_status  {
    struct ofp_action_experimenter_header header; /* 8 bytes, SPTN_EXPERIMENTER_ID */
    uint16_t        exp_type;           /* SPTN_ACT_CHECK_DROP_STATUS */
    uint16_t        type;               /* type of drop status entry */
```



```
    uint32_t        index;              /* index argument */
};
OFP_ASSERT(sizeof(struct sptn_action_check_drop_status) == 16);
```

This Class Based Count and Color Based Count actions require a block index argument. Traffic Class and Color arguments should be programmed to SPTN_CTR_CLASS_ANY or SPTN_CTR_COLOR_ANY in order to have the value supplied by the switch.

```
struct sptn_action_class_based_count  {
    struct ofp_action_experimenter_header header; /* 8 bytes, SPTN_EXPERIMENTER_ID */
    uint16_t        exp_type;           /* SPTN_ACT_CLASS_BASED_COUNT */
    uint8_t         traffic_class;      /* SPTN_CTR_CLASS_ANY */
    uint8_t         pad;                /* align message on 32-bit boundary */
    uint32_t        index;              /* Counter block index argument */
};
OFP_ASSERT(sizeof(struct sptn_action_class_based_count) == 16);

struct sptn_action_color_based_count  {
    struct ofp_action_experimenter_header header; /* 8 bytes, SPTN_EXPERIMENTER_ID */
    uint16_t        exp_type;           /* SPTN_ACT_COLOR_BASED_COUNT */
    uint8_t         color;              /* SPTN_CTR_COLOR_ANY */
    uint8_t         pad;                /* align on 32-bit boundary */
    uint32_t        index;              /* Counter block index argument */
};
OFP_ASSERT(sizeof(struct sptn_action_color_based_count) == 16);
```

9.9 Match Fields

Table 5 lists the added pipeline match fields.

Table 5 Match Fields

Field	Bits	Masked	Prerequisite	Description
Traffic Class	4	No		Pipeline metadata. QoS traffic class.
Color	2	No		Pipeline metadata. Drop precedence. Values are: 00: Green 01: Yellow 10: Red 11: reserved
VLAN DEI	1	No	VLAN tag	Drop eligibility indication from 802.1Q.
QoS Index	8	No	None	Pipeline metadata. Used when looking up Traffic Class and Color based on packet contents in QoS Trust Flow tables.
openFlowMpId	32	No	None	Pipeline metadata. Used to identify a local MEP or MIP instance.

Field	Bits	Masked	Prerequisite	Description
MPLS_TTL	8	No	ETH-TYPE=0x8847	TTL field in MPLS shim header.
MPLS_L2_PORT	32	Yes		Pipeline metadata. Used to identify an MPLS-TP pseudo wire endpoint.
MPLS_DATA_FIRST_NIBBLE	4	No	ETH-TYPE=0x8847 and MPLS_BOS=1	Determine if data (0000b) or control (0001b)
MPLS_ACH_CHANNEL	16	No	MPLS control frame – MPLS DATA FIRST NIBBLE is one	
MPLS_NEXT_LABEL_IS_GAL	1	No	ETH-TYPE=0x8847	Pipeline metadata derived from the packet parser "peeking" at the next label.
OAM_Y1731_MDL	3	No	ETH-TYPE=0x8902	OAM PDU Maintenance Domain Level
OAM_Y1731_OPCODE	8	No	ETH-TYPE=0x8902	OAM PDU opcode
COLOR_ACTIONS_INDEX	32	No	None	Pipeline metadata. Used to identify an entry in the Color Based Actions Flow Table.
TxFCl	64	No		OAM Data Counter Table value set by Set-Counter-Fields. Pipeline metadata field sent to Network Protection App. Read only from Controller.
RxFCl	64	No		OAM Data Counter Table value set by Set-Counter-Fields. Pipeline metadata field sent to Network Protection App. Read only from Controller.
RxTIME	64	No		Timestamp value for current OAM PDU. Pipeline metadata field sent to Network Protection App. Read only from Controller.
Protection_Index	8	No		Indicates whether label is for protection path (0). Other values represent working paths.
MPLS_TYPE	16	No	varies	Bit mask. Indicates flow type: 0: None (default) 1: VPWS all others Reserved

Field	Bits	Masked	Prerequisite	Description
ALLOW_VLAN_TRANSLATION	1	No	None	Used in egress pipeline to enable VLAN translation. Set in L2 Unfiltered Interface group entry type.

The match fields are programmed using the following assignments:

```
/* SPTN Experimenter Match Field types */
enum sptn_match_exp_fields {
    SPTN_OXM_TRAFFIC_CLASS =            2,
    SPTN_OXM_COLOR =                    3,
    SPTN_OXM_DEI =                      4,
    SPTN_OXM_QOS_INDEX =                5,
    SPTN_OXM_OPENFLOWMPID =             6,
    SPTN_OXM_MPLS_TTL =                 7,
    SPTN_OXM_MPLS_L2_PORT =             8,
    SPTN_OXM_MPLS_DATA_FIRST_NIBBLE =   11,
    SPTN_OXM_MPLS_ACH_CHANNEL =         12,
    SPTN_OXM_MPLS_NEXT_LABEL_IS_GAL =   13,
    SPTN_OXM_OAM_Y1731_MDL =            14,
    SPTN_OXM_OAM_Y1731_OPCODE =         15,
    SPTN_OXM_COLOR_ACTIONS_INDEX =      16,
    SPTN_OXM_TXFCL =                    17,
    SPTN_OXM_RXFCL =                    18,
    SPTN_OXM_RX_TIMESTAMP =             19,
    SPTN_OXM_PROTECTION_INDEX           21,
    SPTN_OXM_MPLS_TYPE =                23,
    SPTN_OXM_ALLOW_VLAN_TRANSLATION =   24
}
```

These use the OpenFlow Experimenter OXM TLV definition as follows:

```
struct sptn_oxm_match_field  {
    uint32_t  oxm_header;         /* oxm_class = OFPXMC_ EXPERIMENTER */
    uint32_t  experimenter; /* SPTN Experimenter ID: FF-00-00-0A*/
};
OFP_ASSERT(sizeof(struct sptn_oxm_match_field) == 8);
```

The `oxm_class` field in the header is set to `OFPXMC_EXPERIMENTER` and the `oxm_field` contains a value from `sptn_match_exp_fields`. This can be followed by arguments as specified in [3]. The payload length is given in the `oxm_length` field, so that the total length of the field is `4+oxm_length`. Note that the minimum argument field size is a byte. An OXM TLV does not need to be padded or aligned.

9.10 Extension Fields

This specification uses fields defined in the OpenFlow extensions documents [7].

EXT-244 defines the encoding for Packet Registers.

```
/* Structure for OXM field output match. */
struct onf_oxm_packet_regs  {
    uint32_t      oxm_header; /* oxm_class = OFPXMC_PACKET_REGS,
                                 oxm_field = <N>. */
    uint64_t      value;      /* Packet Register value. */
};
```



```
OFP_ASSERT(sizeof(struct onf_oxm_packet_regs) == 12);
```

EXT-320 defines the encoding for the Copy Field action, reproduced here. Table features encodings are reproduced here and described in detail in the “Copy Field action Extension” document.

```
ONF_EXPERIMENTER_ID = 0x4F4E4600

/* Action types */
enum onf_act_exp_type {
    ONFACT_ET_COPY_FIELD = 3200, /* Copy-Field action. */
};

/* Action structure for ONFACT_ET_COPY_FIELD. */
struct onf_act_copy_field {
    uint16_t type;            /* OFPAT_EXPERIMENTER. */
    uint16_t len;             /* Length is padded to 64 bits. */
    uint32_t experimenter;    /* ONF_EXPERIMENTER_ID. */
    uint16_t exp_type;        /* ONFACT_ET_COPY_FIELD. */
    uint8_t pad2[2];
    uint16_t n_bits;          /* Number of bits to copy. */
    uint16_t src_offset;      /* Starting bit offset in source. */
    uint16_t dst_offset;      /* Starting bit offset in destination. */
    uint8_t pad[2];           /* Align to 32 bits. */

/* Followed by:
    * - Exactly 8, 12 or 16 bytes containing the oxm_ids, then
    * - Enough all-zero bytes (either 0 or 4) to make the action a whole
    * multiple of 8 bytes in length */

    uint32_t oxm_ids[0];      /* Source and destination OXM headers */
};
OFP_ASSERT(sizeof(struct ofp_action_copy_field) == 20);


/* Action types */
enum onf_tfp_exp_type {
    ONFTFP_ET_WRITE_COPYFIELD = 3200, /* Write Copy-Field property. */
    ONFTFP_ET_APPLY_COPYFIELD = 3201, /* Apply Copy-Field property. */
};
```

The table feature properties `ONFTFP_ET_WRITE_COPYFIELD` and `ONFTFP_ET_APPLY_COPYFIELD` use the following structure.

```
/* Table Feature Property structure for ONFTFP_ET_WRITE_COPYFIELD
            * and ONFTFP_ET_APPLY_COPYFIELD. */
struct onf_tfp_copy_field {
    uint16_t type;            /* One of OFPTFPT_EXPERIMENTER,
                              OFPTFPT_EXPERIMENTER_MISS. */
    uint16_t length;          /* Length in bytes of this property. */
    uint32_t experimenter;    /* ONF_EXPERIMENTER_ID. */
    uint32_t exp_type;        /* ONFTFP_ET_WRITE_COPYFIELD or
                              ONFTFP_ET_APPLY_COPYFIELD. */
/* Followed by:
    * - Exactly (length - 4) bytes containing the oxm_ids, then
```



```
    * - Exactly (length + 7)/8*8 - (length) (between 0 and 7)
    * bytes of all-zero bytes */

    uint32_t oxm_ids[0];      /* Array of OXM headers */
};
OFP_ASSERT(sizeof(struct onf_tfp_copy_field) == 12);
```

EXT-233 defines an Experimenter encoding for ACTSET_OUTPUT. This uses the ONF Experimenter id: 0x4F4E4600.

```
/* OXM types */
enum onf_oxm_exp_type {
   ONFOXM_ET_ACTSET_OUTPUT = 43, /* Output port from action set Metadata. */
};
```

10 SPTN TTP

The accompanying text file, “MPLS-TP-openflow-protocol-extensions-for-SPTN.json”, contains the normative JSON TTP.

Figure 31 illustrates the complete TTP pipeline. This diagram should be considered informative.

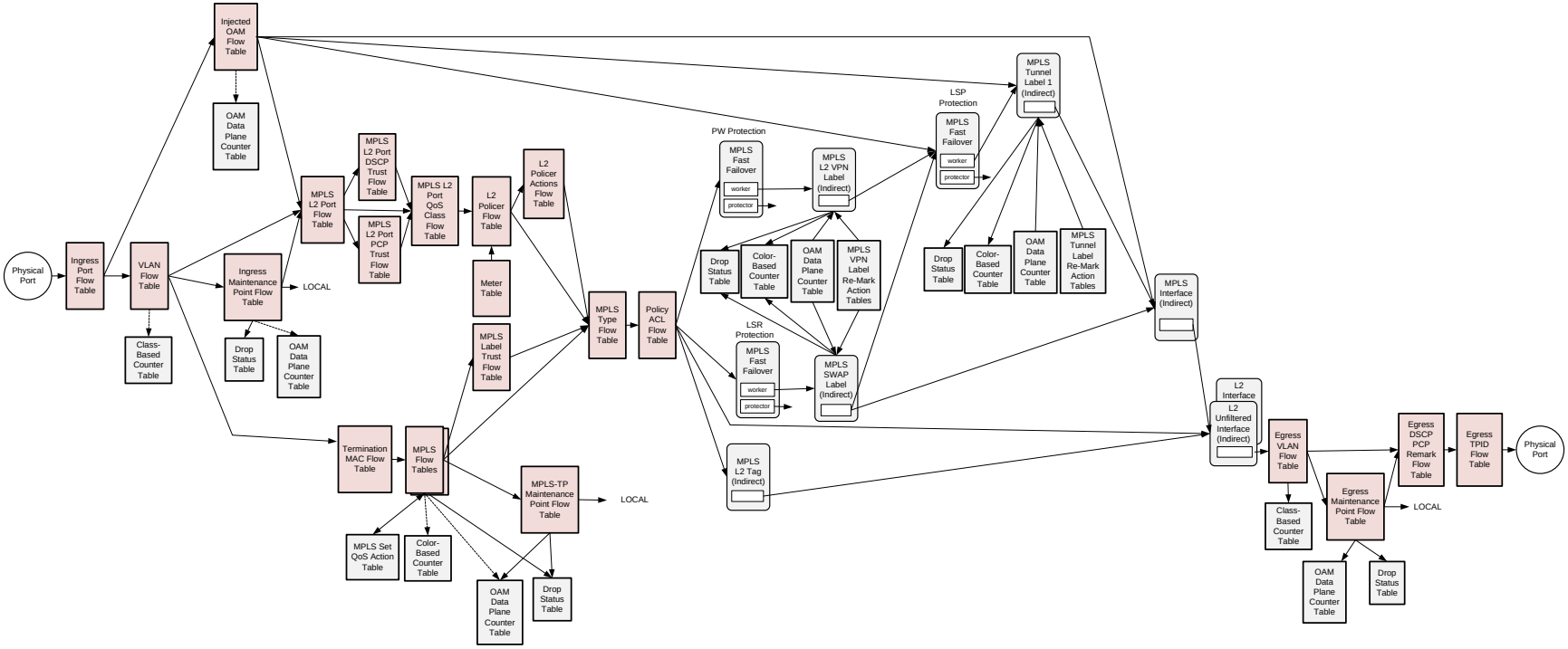

Figure 31 Complete TTP Pipeline

11 YANG Model

The accompanying text files, “of-config-1.2-sptn-oam.yang” and “of-config-1.2-sptn-qos.yang”, contain the normative YANG augmentations to OF-CONFIG 1.2 for OAM and QoS.

12 References

[1] OpenFlow 1.3.1 Specification, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf

[2] OpenFlow Management and-Configuration Protocol 1.2 Specification, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/ openflow-config/of-config-1.2 pdf

[3] OpenFlow 1.3.4 Specification, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf

[4] OpenFlow 1.4.1 Specification, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.4.1.pdf

[5] OpenFlow 1.5.1 Specification, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf

[6] OpenFlow Table Type Patterns 1.0, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/OpenFlow Table Type Patterns 0v1.0.pdf

[7] OpenFlow Extensions Package, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-extensions-1.3.x-package.zip

[8] Pffaf, B., and Davie, B, “The Open vSwitch Database Management Protocol,” RFC7047, December 2013

[9] “Technique Specification for CMCC Packet Transport Network,” version 1.0.0, CMCC Telecom Enterprise Standard.

[10] “White Paper of Super PTN Technology,” version 0.1, China Mobile Research Institute

[11] “Technical Specification for CMCC Enterprise Customer Accessing Packet Transport Network,” version 1.0, CMCC Telecom Enterprise Standard

[12] Allan, D.,Swallow, G., and Drake, J., “Proactive Connectivity Verification, Continuity Check, and Remote Defect Indication for the MPLS Transport Profile,” RFC 6428, November, 2011

[13] International Telecommunications Union, "Operations, administration and maintenance mechanism for MPLS-TP in packet transport networks", Recommendation ITU-T G.8113.1/Y.1372.1, November, 2012

[14] International Telecommunications Union, "Operations, administration and maintenance mechanism for MPLS-TP networks using the tools defined for MPLS", Recommendation ITU-T G.8113.2/Y.1372.2, November, 2012

[15] International Telecommunications Union, “OAM functions and mechanisms for Ethernet based networks,” Recommendation ITU-T G.8013/Y.1731, November, 2013
[16] Heinanen, J. and Guerin, R., “A Single Rate Three Color Marker,” RFC 2697, September, 1999
[17] Heinanen, J. and Guerin, R., “A Two Rate Three Color Marker,” RFC 2698, September, 1999
[18] Aboul-Magd, O., Rabie, S., “Differentiated Service Two-Rate, Three-Color Marker with Efficient Handling of in-Profile Traffic,” RFC 4115, July, 2005
[19] Vigoureux, M., Ward, D., and Betts, M., “Requirements for Operations, Administration, and Maintenance (OAM) in MPLS Transport Networks,” RFC 5860, May, 2010
[20] IEEE Standard for Local and metropolitan area networks – Virtual Bridged Local Area Networks – Amendment 5: Connectivity Fault Management, IEEE Std 802.1ag™-2007
[21] IEEE Standard for Local and metropolitan area networks – Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Amendment: Media Access Control Parameters, Physical Layers, and Management Parameters for Subscriber Access Networks, IEEE Std 802.3ah™-2002
[22] International Telecommunications Union, "Characteristics of Ethernet transport network equipment functional blocks", Recommendation ITU-T G.8021/Y.1341, May, 2012
[23] Bjorklund, M., “YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF),” RFC 6020, October, 2010
[24] International Telecommunications Union, "Ethernet Ring Protection Switching", Recommendation ITU-T G.8032/Y.1344, February, 2012
[25] International Telecommunications Union, "Ethernet linear protection switching", Recommendation ITU-T G.8031/Y.1342, June, 2011
[26] International Telecommunications Union, "Linear protection switching for MPLS transport profile", Recommendation ITU-T G.8131/Y.1382, July, 2014
[27] Lasserre, M., and Kompella, V., “Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling,” RFC 4762, January, 2007
[28] https://rs.opennetworking.org/wiki/display/PUBLIC/ONF+Registry

13 CONTRIBUTORS

Weiqiang Cheng	China Mobile
Lei Wang	China Mobile
Joseph Tardo	Broadcom Limited
Zongying He	Broadcom Limited
Yanguang Qu	Broadcom Limited
Tao Lang	Microsemi
Desheng Sun	ZTE
Shengbo (Steve) Fu	Fiberhome
Lynn Lu	Ericsson