# 6 VLC sublayer

## 6.1 VLC Functional Block Diagram

## 6.2 VLC Classification and Translation Engine

### 6.2.1 *CTE rule* structure

#### 6.2.1.1 *CTE rule* classification conditions

##### 6.2.1.1.1 Comparison operators

##### 6.2.1.1.2 Classification fields

The CTE comparison operation elements recognize the fields shown in Table 6-2. ~~Note that field IDs listed below represent unique identifiers of various fields accessible to the *CTE rules*. The field IDs are shown in all capital letters as opposed to the field names, which are shown as a mixture of capital and lowercase letters.~~

**Table 6-12—Classification fields**

| ~~FIELD_ID~~Field Name | Field size (bits) | Description |
|---|---|---|
| *DstAddr*~~FID_DST_ADDR~~ | 48 | This field ~~ID identifies~~represents the outermost MAC destination address~~(*DstAddr*) field~~. |
| *SrcAddr*~~FID_SRC_ADDR~~ | 48 | This field represents~~ID identifies~~ the outermost MAC source address ~~(*SrcAddr*) field~~. |
| *Ethertype*~~FID_LEN_TYPE~~ | 16 | This field ~~ID identifies~~represents the ~~outermost~~ Ethernet Type/Length ~~(*Ethertype*) fiel~~value~~d~~, per IEEE Std 802.3, 3.1.1 |
| *Vlan0*~~FID_VLAN0~~ | 32 | This field represents~~This field ID identifies~~ the first (outermost) VLAN tag ~~(*Vlan0*)~~ field following the *SrcAddr* field. If no VLAN tags follow the *SrcAddr* field, then the *Vlan0* field does not exist. |
| *Vlan1*~~FID_VLAN1~~ | 32 | This field represents~~This field ID identifies~~ the innermost VLAN tag (i.e., the VLAN tag that follows the outermost *Vlan0* field). If no VLAN tags follow the *Vlan0* field, then the *Vlan1* field does not exist. |
| *Subtype*~~FID_SUBTYPE~~ | 8 | ~~This field ID identifies the *Subtype* field.~~ This field represents an octet immediately following the *Ethertype* field, regardless of whether the frame format associated with this Ethertype includes any actual subtype field or not.<br>The *Subtype* field in VLCPDUs is defined in 5.2. An example of this field in non-VLCPDU is the *Subtype* field in Slow Protocol PDUs (see IEEE Std 802.3, 57A.4). |
| *xPduDstAddr*~~FID_XPDU_DST_ADDR~~ | 48 | This field represents~~This field ID identifies~~ the MAC destination address ~~(*xPduDstAddr*)~~ field of an xPDU carried within the VLCPDU payload. This field exists only in VLCPDUs with the value of the *Subtype* field equal to `SUBTYPE_L2`. |
| *xPduSrcAddr*~~FID_XPDU_SRC_ADDR~~ | 48 | This field represents~~This field ID identifies~~ the MAC source address ~~(*xPduSrcAddr*)~~ field of an xPDU carried within the VLCPDU payload. This field exists only in VLCPDUs with the value of the *Subtype* field equal to `SUBTYPE_L2`. |
| *xPduEthertype*~~FID_XPDU_TYPE_LEN~~ | 16 | This field represents~~This field ID identifies~~ the Ethertype (*xPduEthertype*) field of an xPDU carried within the VLCPDU payload. This field exists only in VLCPDUs with the value of the *Subtype* field equal to `SUBTYPE_L2` or `SUBTYPE_EPD`. |
| *xPduVlan0*~~FID_XPDU_VLAN0~~ | 32 | This field represents ~~This field ID identifies~~ the first (outermost) VLAN tag ~~(*xPduVlan0*)~~ field of an xPDU carried within the VLCPDU payload. This field exists only in VLCPDUs with the value of the *Subtype* field equal to `SUBTYPE_L2` or `SUBTYPE_EPD`. |
| *xPduVlan1*~~FID_XPDU_VLAN1~~ | 32 | This field represents ~~This field ID identifies~~ the ~~first~~ second (innermost) VLAN tag ~~(*xPduVlan1*)~~ field of an xPDU carried within the VLCPDU payload. This field exists only in VLCPDUs with the value of the *Subtype* field equal to `SUBTYPE_L2` or `SUBTYPE_EPD` and only if the xPDU carried in the VLCPDU payload contains the innermost VLAN tag. |

| | | |
|---|---|---|
| *xPduSubtype*~~F ID_XPDU_SUBTYPE~~ | 8 | This field represents ~~This field ID identifies~~ the subtype ~~(*xPduSubtype*)~~ field of an xPDU carried within the VLCPDU payload. This field exists only in VLCPDUs with the value of the *Subtype* field equal to SUBTYPE_L2 or SUBTYPE_EPD.<br>The *xPduSubtype* field represents an octet immediately following the *xPduEthertype* field, regardless of whether the frame format associated with this Ethertype includes any actual subtype field or not. |

### 6.2.1.2 *CTE rule* modification actions

An action represents a specific modification of a single header field. A field may be modified using any of the atomic operations defined in Table 6-3.

**Table 6-~~23~~—Actions used in *CTE rule*s**

| Action | Numeric Code | Mnemonic / Description |
|---|---|---|
| Add a field | 0xAD | ADD(taget_field~~_id~~, field_value)<br>This operation adds a field ~~of the type indicated by the~~ taget_field~~_id~~ ~~and~~ having the value of field_value. |
| Remove (delete) a field | 0xDE | REMOVE(taget_field~~_id~~)<br>This operation removes a field ~~of the type indicated by the~~ taget_field~~_id~~. The result of the REMOVE operation is undefined if the field ~~indicated by the~~ taget_field~~_id~~ is not present in the frame. |
| Replace (change) a field | 0xCE | REPLACE(taget_field~~_id~~, field_value)<br>This operation replaces the value of the field ~~indicated by the~~ taget_field~~_id~~ with the value of field_value. The result of the REPLACE operation is undefined if the field ~~indicated by the~~ taget_field~~_id~~ is not present in the frame. |
| Copy (duplicate) a field | 0xD8 | COPY(taget_field~~_id~~, source_field~~_id~~)<br>This operation adds a field ~~of the type indicated by the~~ taget_field~~_id~~ with the same value as ~~value of~~ the field ~~indicated by the~~ source_field~~_id~~. The result of the COPY operation is undefined if the field ~~indicated by the~~ taget_field~~_id~~ is already present in the frame or if the field ~~indicated by the~~ source_field~~_id~~ is not present in the frame. The result is also undefined if the ~~fields identified by the~~ taget_field~~_id~~ and source_field~~_id~~ are not of the same size. |

The actions are applied in the order they are listed in the rule. The list of modifiable fields is shown in Table 6-2, with the following exceptions:

- No modification actions shall be applied to the FID_SRC_ADDR field;
- Only REPLACE action may be applied to the FID_DST_ADDR and FID_LEN_TYPE fields.

Note that in a double-tagged frame, deleting an outermost VLAN tag produces a frame with an outermost VLAN tag only. Therefore, applying the following two commands results in an error:

`REMOVE(FID_VLAN0)`
`REMOVE(FID_VLAN1)` – error: FID_`VLAN1` field does not exists

However, any of the following two sequences of actions achieve the desired result of removing both VLAN tags:

`REMOVE(FID_VLAN0)` – delete outermost tag first
`REMOVE(FID_VLAN0)` – delete the remaining tag

`REMOVE(FID_VLAN1)` – delete innermost tag first
`REMOVE(FID_VLAN0)` – delete the remaining tag

# 7 Protocol-Specific behavior

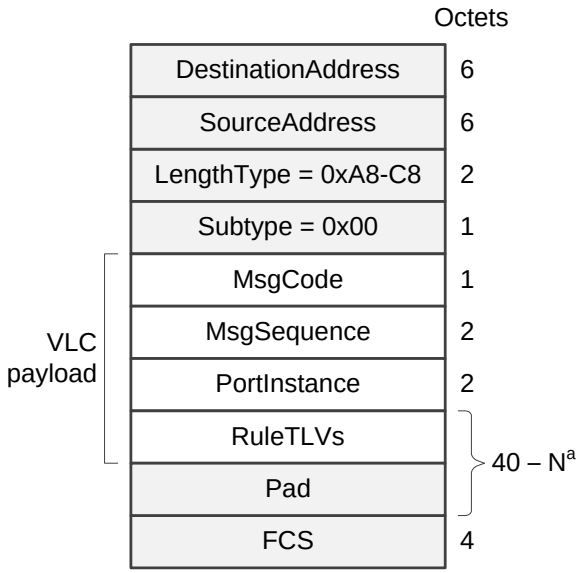
# 8 VLC Management

## 8.1 VLC Configuration

The tunnels originate and terminate in the VLC-aware devices. The tunnels are configured by means of provisioning specific *CTE rule*s for the tunnel entry and exit points. These rules are provisioned by the operator using the *VLC_CONFIG* VLCPDUs, which carry a set of *condition-encoding* TLVs and a set of *action-encoding* TLVs.

### 8.1.1 Configuration VLCPDU

The *VLC_CONFIG* VLCPDU format shall be as depicted in Figure 8-1. The *VLC_CONFIG* VLCPDU is used as both a request to configure a *CTE rule* as well as a response containing the result of the configuration request.




a – Maximum field length depends on frame type (see Figure 5-1).

**Figure 8-1—*VLC_CONFIG* VLCPDU format**

The *VLC_CONFIG* VLCPDU is an instantiation of the generic VLCPDU (see Figure 5-1). It is identified by the *Subtype* field value of 0x00. The structure of the *VLC payload* is defined as follows:

—*MsgCode*:

The *MsgCode* field identifies whether the *VLC_CONFIG* message is a request message or a response. If the VLCPDU is a request, this field encodes the requested action. If the VLCPDU is a response, this field echoes the requested action and encodes the result code for this action. The format of the *MsgCode* field is shown in Table 8-1.

**Table 8-1—Format of the *MsgCode* field**

| Bits | Field name | Value | Description |
|---|---|---|---|
| 3:0 | *MsgType* | 0x0 | The message is a request |
| | | 0x1 | The message is a response indicating successful action |

| | | | |
|---|---|---|---|
| | | 0x2 | The message is a response indicating failed action |
| | | 0x3 | The message is a response indicating that no action was necessary |
| | | 0x4 | The message is a response indicating invalid request |
| | | 0x5 to 0xF | Reserved, ignored on reception |
| 7:4 | *RequestCode* | 0x0 | Query all rules |
| | | 0x1 | Add a rule |
| | | 0x2 | Remove a rule |
| | | 0x4 to 0xF | Reserved, ignored on reception |

—*MsgSequence*:

In situations when a VLC configuration request or a response consists of multiple messages, this field identifies the message sequence number. The format of the *MsgSequence* field is shown in Table 8-2.

**Table 8-2—Format of the *MsgSequence* field**

| Bits | Field name | Value | Description |
|---|---|---|---|
| 14:0 | *MsgCounter* | 0x00-01 to 0x7F-FF | A counter that increments by one for each message in a sequence. In the first message in a sequence, the *MsgCounter* is equal to 1. |
| 15 | *EndOfSequence* | 0 | This message is not the last message in a sequence |
| | | 1 | This message is the last message in a sequence |

When a request or a response consists of a single VLCPDU, the *MsgCounter* subfield is equal to 0x00-01 and the *EndOfSequence* flag is equal to 1.

Note that even when a VLC configuration request or a response consists of multiple messages, a single rule is not split across multiple messages and as such – no reassembly mechanism is necessary to reconstruct any rule. An example scenario where the response consists of multiple messages would be a VLC configuration response to a ‘Query all rules’ request, where multiple rules are being reported.

—*PortInstance*:

This field identifies a port instance in the VLC-aware device to which the given *VLC_CONFIG* VLCPDU applies. The format of the *PortInstance* field is shown in Table 8-3.

**Table 8-3—Format of the *PortInstance* field**

| Bits | Field name | Value | Description |
|---|---|---|---|
| 14:0 | *PortIndex* | 0x00-00 to 0x7F-FF | Index of a port (VLC sublayer) to which the requested action is to be applied. |
| 15 | *Direction* | 0 | The rule is to be applied to the transmit path of VLC sublayer (i.e., an egress rule) |
| | | 1 | The rule is to be applied to the receive path of VLC sublayer (i.e., an ingress rule) |

In the VLC response message, this field reflects the *PortInstance* field value from the corresponding VLC request message.

—*RuleTLVs*:

This field includes one or more *CTE rule* TLV(s) as defined in 8.1.2. The combined size of the *RuleTLV* and *Pad* fields ranges between 40 and *N*, where *N* is defined in Figure 5-1.

### 8.1.2 *CTE rule* TLV structure

The structure of a *CTE rule* TLV is shown in Table 8-4. Each *VLC_CONFIG* VLCPDU shall contain at least one *CTE rule* TLV.

**Table 8-4—*CTE rule* TLV structure**

| Field Size (octets) | Field Name | Value | Description |
|---|---|---|---|
| 1 | *Type* | 0xC0 | Type code identifying the condition-encoding TLV |
| | | 0xAC | Type code identifying the action-encoding TLV |
| | | 0x00 | Type code indicating that there are no more TLVs to process. The Length field and other fields (if present) are ignored. The TLV with Type = 0x00 shall be the last TLV in every *VLC_CONFIG* VLCPDU and it may be the only TLV in the *VLC_CONFIG* VLCPDU. |
| 1 | *Length* | *V*+*M*+4 | The *Length* field encompasses the entire TLV, including the *Type* and *Length* fields. A TLV with length of 0x00 through 0x03 is invalid. |
| 1 | *Operation* [a] | per Table 6-1 | Comparison operator code, if the TLV *Type* = 0xC0 |
| | | per Table 6-3 | Action code, if the TLV *Type* = 0xAC |
| *1* | ~~*FieldCode*~~ *FieldId* [a] | per Table 6-2 | Identifies a field to be used in a comparison, or to be modified by an action. |
| *V* | *Value* | Various | The value to be used in a comparison or by an Add/Change action. Some TLVs may omit this field. |
| *M* [b] | *Mask* | various | The mask pattern to be used in a comparison condition. The mask pattern is applied as a bitwise-AND operation to both the value to be used in a comparison (see the *Value* field above) as well the value of the field identified by the *FieldCode* parameter of this TLV. Some TLVs may omit this field [c]. When *Mask* is omitted, the comparison applies to the entire field. |

[a] Fields *Operation* and ~~*FieldCode*~~ *FieldId* shall be present in all TLVs, even if they are not used. When these fields are not used, they should be set to the value of zero.

[b] The length *M* of *Mask* field shall be the same as the length of *Value* field, if mask field is present. Otherwise, the length *M* is considered to be equal to zero.

[c] If a *CTE rule* TLV omits the *Value* field, the *Mask* field shall also be omitted.

The subfield *FieldId* carries an identification code for one of the fields defined in Table 6-2. The available identification codes are shown in Table 8-5.

**Table 8-5—Subfield *FieldId* values**

| Field Name | *FieldId* value | Field Name | *FieldId* value |
|---|---|---|---|
| *DstAddr* | 0x01 | *xPduDstAddr* | 0x11 |
| *SrcAddr* | 0x02 | *xPduSrcAddr* | 0x12 |
| *Ethertype* | 0x03 | *xPduEthertype* | 0x13 |
| *Vlan0* | 0x04 | *xPduVlan0* | 0x14 |
| *Vlan1* | 0x05 | *xPduVlan1* | 0x15 |
| *Subtype* | 0x06 | *xPduSubtype* | 0x16 |