| | |
|---|---|
| **Task Group** | Data Center Bridging |
| **Revision** | 1.01 |
| **Author** | Manoj Wadekar (Qlogic), et al |

# DCB Capability Exchange Protocol Base Specification

# Rev 1.01

# Modification History

| Rev | Originator | Comment |
|---|---|---|
| 1.0 | Manoj Wadekar | Initial Submitted Version |
| 1.01 | Manoj Wadekar | Added Application TLV |

# TABLE OF CONTENTS

# LIST OF TABLES



# LIST OF FIGURES



# Terminology

| Term | Description |
|---|---|
| BCN | Backward Congestion Management |
| CM | Congestion Management |
| DCB | Data Center Bridging |
| DCBX | DCB Capability Exchange Protocol |
| LLDP | Link Layer Discovery Protocol, IEEE802.1AB |
| LLDPDU | An LLDP PDU |
| NIC | Network interface controller |
| OS | Operating System. |
| OUI | Organizationally Unique Identifier |
| PDU | Protocol Data Unit |

| Term | Description |
|---|---|
| PFC | Priority-based Flow Control (same as Per Priority Pause or Class Based Flow Control) |
| PG | Priority Groups |
| RX | Receive |
| SNMP | Simple Network Management Protocol |
| TLV | Type Length Value |
| TTL | Time to Live |
| TX | Transmit |

# Related Documents

| DCB Feature Specifications |
|---|
| Proposal for Priority Based Flow Control<br>http://www.ieee802.org/1/files/public/docs2008/bb-pelissier-pfc-proposal-0508.pdf<br>Packet Scheduling with Priority Grouping and Bandwidth Allocation for DCB Networks<br>http://www.ieee802.org/1/files/public/docs2008/az-wadekar-ets-proposal-0608-v1.01.pdf |

# 1. Authors

The following people, with company affiliations, have contributed to the preparation of this proposal:

Amit Shukla – Juniper
Anoop Ghanwani - Brocade
Anjan – Cisco
Anthony Faustini - Cisco
Asif Hazarika – Fujitsu
Awais Nemat – Marvell
Bruce Klemin – Qlogic
Brice Kwan - Broadcom
Claudio DeSanti- Cisco
Craig W. Carlson - QLogic
Dan Eisenhauer – IBM
Danny J. Mitzel - Brocade
David Peterson – Brocade
Diego Crupniokoff – Mellanox
Dinesh Dutt - Cisco
Douglas Dreyer - IBM
Ed McGlaughlin – Qlogic
Eric Multanen - Intel
Gaurav Chawla - Dell
Glenn - Brocade
Hemal Purohit - QLogic
Hugh Barrass – Cisco
Ilango Ganga - Intel
Irv Robinson - Intel
J. R. Rivers – Cisco
Jeelani Syed - Juniper
Jeffrey Lynch - IBM
Jim Larsen - Intel
Joe Pelissier - Cisco
John Hufferd – Brocade
John Terry – Brocade
Krishna Doddapaneni - Cisco
Manoj Wadekar – Qlogic
Menu Menuchehry - Marvell
Mike Ko – IBM
Mike Krause - HP
Parag Bhide - Emulex
Pat Thaler - Broadcom
Ravi Shenoy - Emulex
Renato Recio - IBM
Robert Snively - Brocade
Roger Hathorn - IBM

Sanjaya Anand – Qlogic
Sanjay Sane – Cisco
Shreyas Shah - PLX
Silvano Gai - Cisco
Stuart Berman - Emulex
Suresh Vobbilisetty - Brocade
Taufik Ma - Emulex
Uri Elzur - Broadcom

# 2. Introduction

This document details the data center discovery and capability exchange protocol (DCBX) that is used by DCB devices to exchange configuration information with directly connected peers. The protocol may also be used for misconfiguration detection and for configuration of the peer.

This document describes the base protocol which comprises a control state machine and a generic feature state machine. For each feature that is to be supported by DCBX, the following information must be provided:

- The parameters to be exchanged;
- How the parameters are used for detecting misconfiguration;
- What action needs to be taken when an error is detected;

This document also lists the above information for the following features:

- Priority Groups (PG)
- Priority-based Flow Control (PFC)

In future, it is likely that additional features may be added to DCBX.

## 2.1 Goals

The following lists the goals of DCBX.

**Discovery of DCB capability in a peer**: DCBX is used to know about the capabilities of the peer device. It is a means to know if the peer device supports a particular feature such as Priority Groups (PG) or Priority-based Flow Control (PFC). For example, it can be used to determine if two link peer devices support PFC.

**DCB feature misconfiguration detection**: DCBX can be used to detect misconfiguration of a feature between the peers on a link. Misconfiguration detection is feature-specific because some features may allow asymmetric configuration.

**Peer configuration of DCB features**: DCBX can be used by a device to perform configuration of DCB features in its link peer. The goal is to provide basic peer to peer configuration through DCBX in the initial version. Future versions of DCBX or another higher layer application can build on top of this to provide more complex configuration distribution mechanisms.

Figure 1 shows a deployment scenario for a network that is using DCBX. DCBX capable links exchange DCB capability and configuration, and conflict alarms are sent to the appropriate management stations. As an example, a boundary is shown indicating which devices support PFC and which do not.

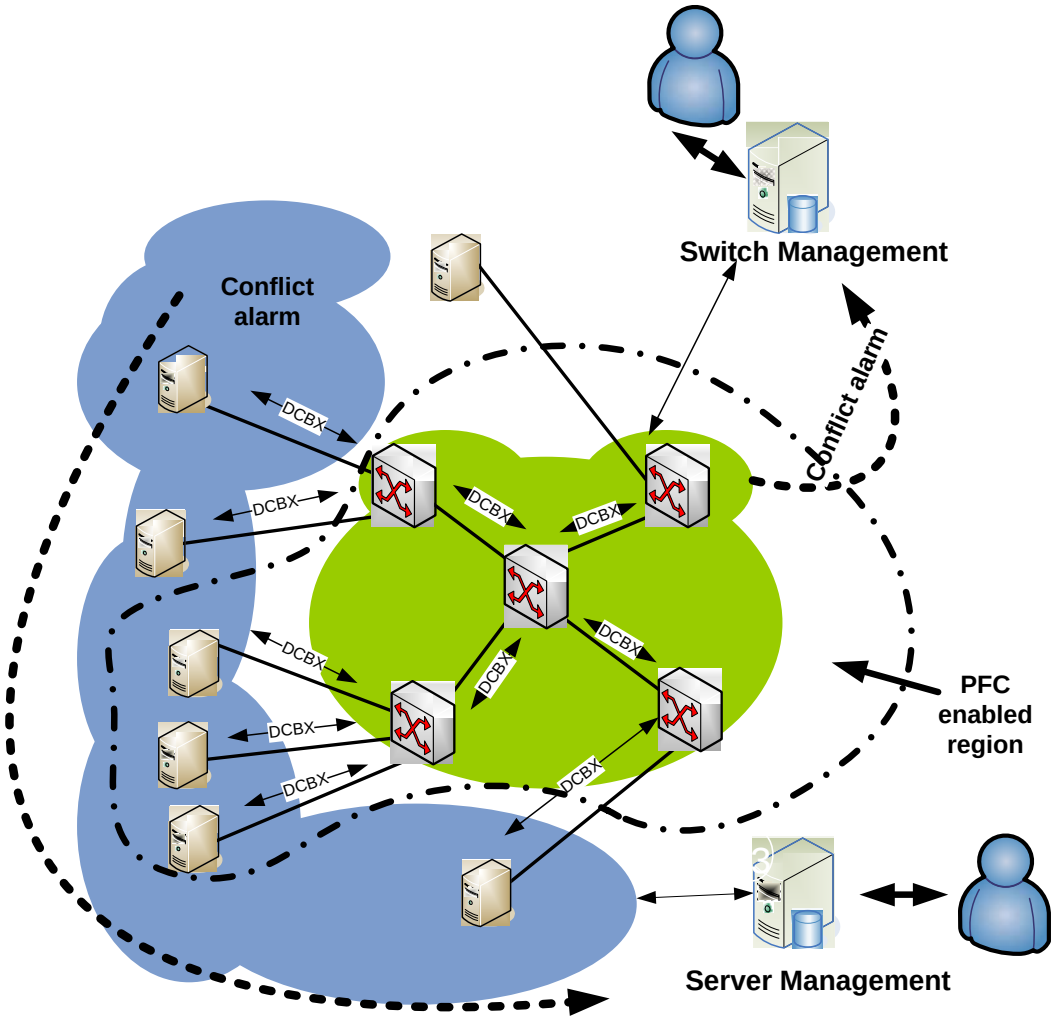


**Figure 1 - DCBX Deployment Scenario**

## 2.2 Types of DCB Parameters

Each DCB feature has a set of parameters. DCB parameters are classified into two broad categories:

- **Exchanged parameters**: Exchanged parameters are sent to the peer. Within these parameters, there are two sub-groups:
  1. Administered parameters: These are the configured parameters.
  2. Operational parameters: This is the operational state of the related administered parameter. Operational state might be different than the administrative/configured state, primarily as a result of the DCBX exchange with the peer. Operational parameters accompany only those administered parameters where there is a possibility that the operational state is different from what was set by their administrator. The operational parameters are included in the the LLDP message only for informational purposes. It might be used by a device to know what is the current operational state of the peer.
- **Local parameters**: Local parameters are not exchanged in LLDP messages.

Figure 2 shows the exchanged parameters that are sent to each peer via LLDP messages.

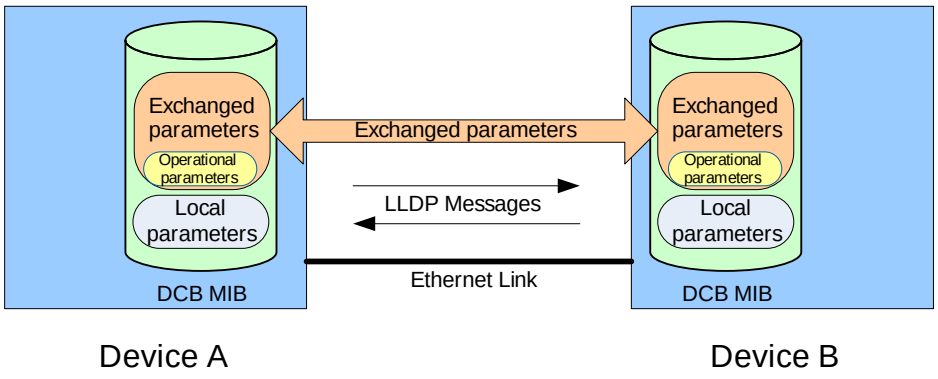


**Figure 2 - Types of Parameters**

## 2.3 DCBX and LLDP

DCBX uses Link Layer Discovery Protocol (LLDP) to exchange parameters between two link peers. LLDP is a unidirectional protocol. It advertises connectivity and management information about the local station to adjacent stations on the same IEEE 802 LAN.

LLDP PDUs carry Type Length Values (TLVs) classified as

1. Mandatory TLVs: Chassis ID, Port ID, TTL, End of LLDPDU.
2. Optional TLVs: Basic Management, 802.1 and 802.3 Organizationally Specific.

DCB exchanged parameters are packaged into Organizationally Specific TLVs. The OUI used for the DCBX TLV is 0x001B21 (IEEE OUI will be used when it becomes available).

Depending on the amount of data required for all features, one or more TLVs, with different sub-types, are defined for DCBX. Within the DCBX TLVs, sub-TLVs are defined for each feature carried by that TLV.

A device capable of any DCB feature must have DCBX enabled by default with an option for DCBX to be administratively disabled.

DCBX is expected to operate over a point to point link. If multiple LLDP neighbors are detected, then DCBX behaves as if the peer's DCBX TLVs are not present until the multiple LLDP neighbor condition is no longer present. An LLDP neighbor is identified by its logical MAC Service Access Identifier (MSAP). The logical MSAP is a concatenation of the chassis ID and port ID values transmitted in the LLDPDU.

LLDP gives administrator control to enable/disable the protocol independently on the Rx side and Tx side. Since DCBX is an acknowledged protocol which uses LLDP, for the protocol to operate correctly both LLDP Rx and Tx must be enabled on the interface on which DCBX runs. The behavior of DCBX is as follows with respect to LLDP Rx/Tx admin state controls:

* If either of Rx or Tx is in disable state, DCBX is disabled on the interface. Neither the control nor feature state machines should run. The LLDPDU's that are generated from this interface do not have any DCBX TLVs. If the peer sends DCBX TLVs they should be ignored as far as the DCBX state machines are concerned.

* When DCBX is currently running and LLDP TX is disabled, then according to the LLDP specification, a shutdown LLDPDU is sent. When the peer receives this PDU, DCBX is determined to be disabled on the peer. This is equivalent to DCBX TLV TTL expired in the Control State machine and Rx.Feature.present() = FALSE in the Feature state machine. If for some reason this frame is lost, then DCBX depends on standard rxInfoTTL expiry of the peer's LLDP TLV's.

* When DCBX is currently running and LLDP Rx is disabled, then all DCBX TLV's including the control TLV should be withdrawn from the LLDP PDUs that the interface generates. The peer's behavior should be the same as discussed in the previous case.

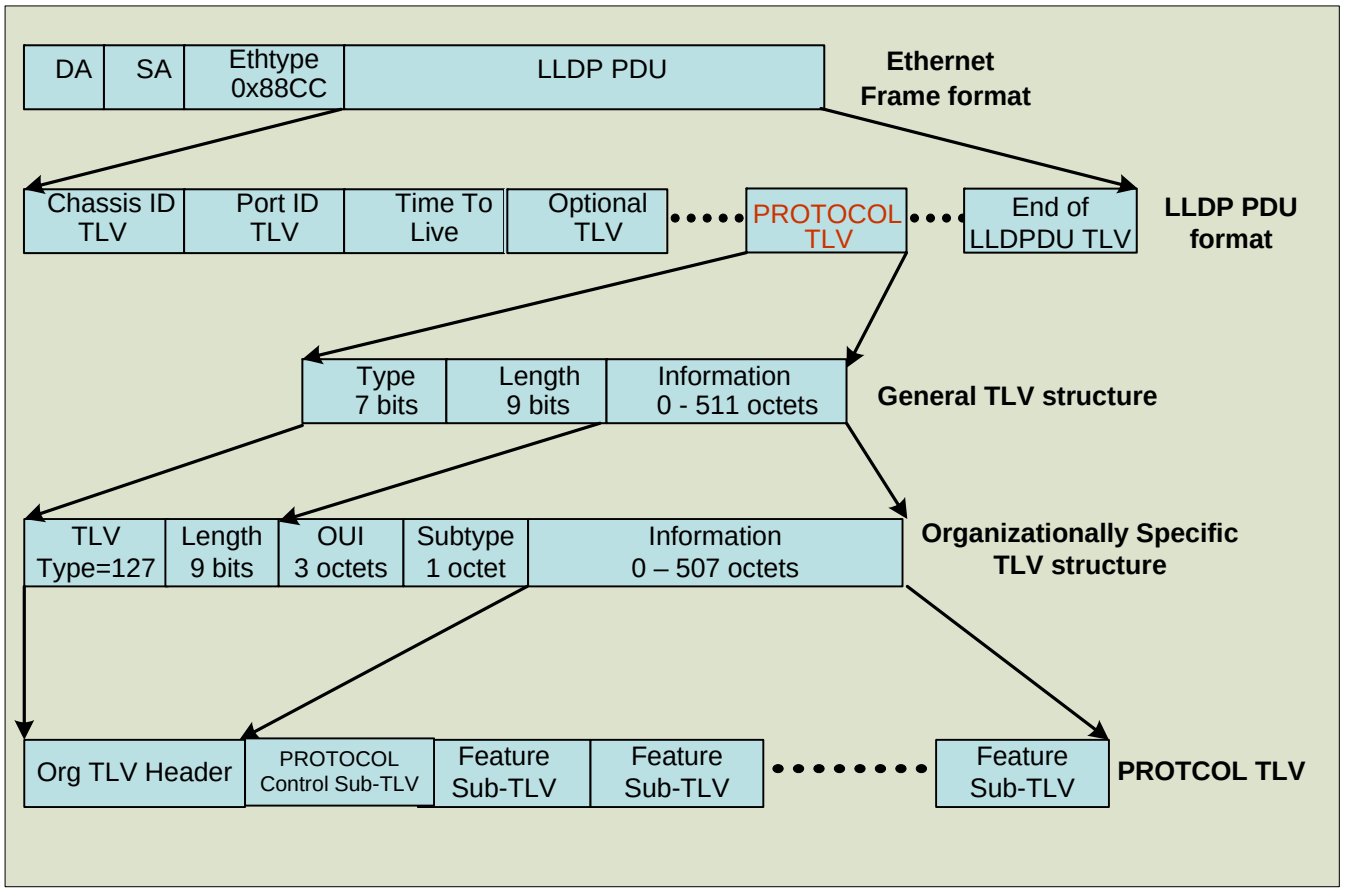


**Figure 3 - LLDP Frame Format**

## 2.3.1 LLDP Modifications

This section lists the proposed modifications to the LLDP protocol for use with DCBX. IEEE 802.1AB REV project is currently working on these modifications. Once standard is published for IEEE 802.1AB-REV, DCBX specification will be appropriately modified.

### 2.3.1.1 Fast initial LLDP Transmissions

The current LLDP protocol can result in a long delay before DCB parameters are exchanged and synchronized.

LLDP transmits a frame after:

- Transmit countdown timer expiration (recommended default value = 30 seconds) and txDelay expiration, OR
- A condition (status or value) change in one or more objects in LLDP local system MIB. (LLDP local system MIB contains only the objects that are sent in LLDP frames. This MIB could be merely a subset of a larger MIB).

After initialization, an LLDP frame is transmitted (considering the initialization as a status change).

An initial LLDP message might not be received by the peer due to different times at which their initialization completes (see Figure 4).

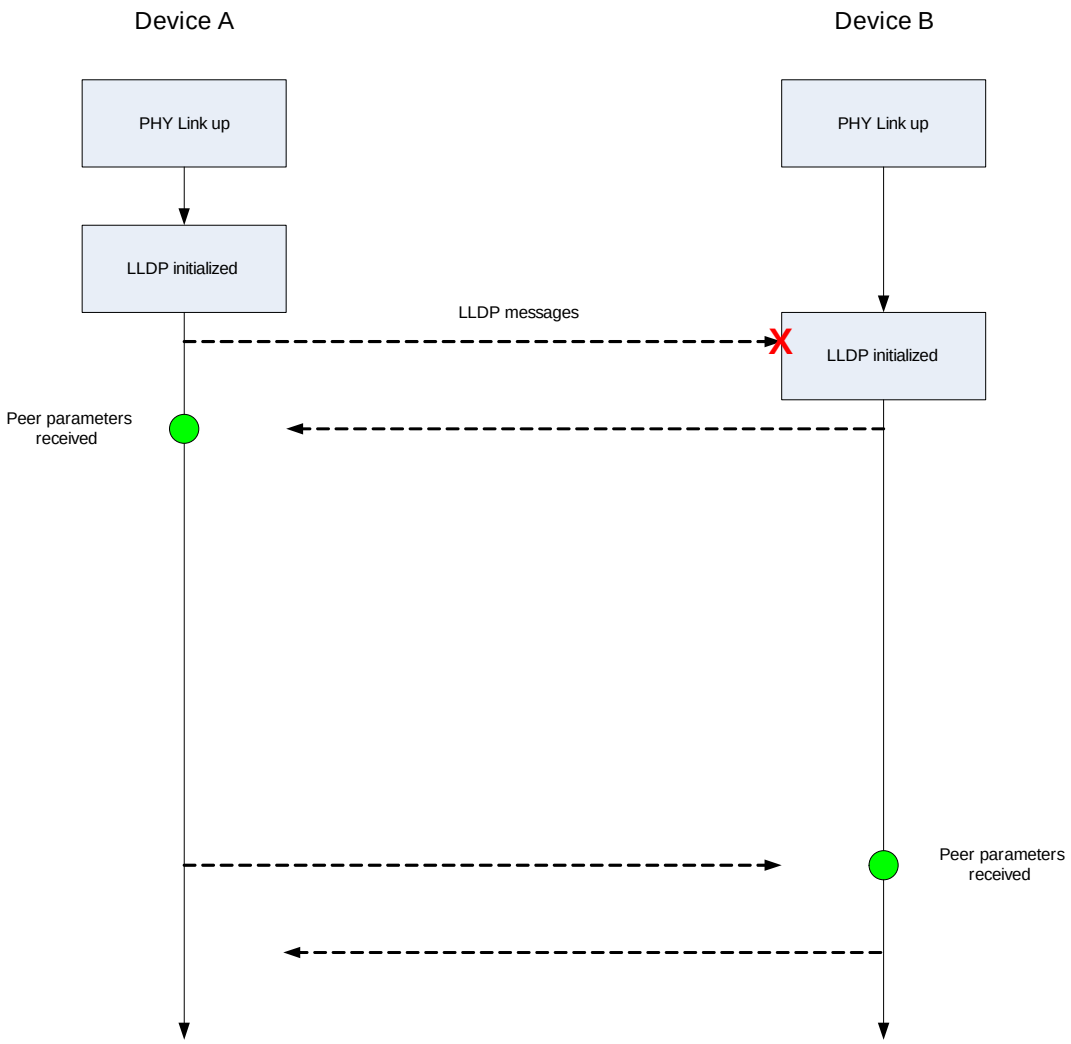


**Figure 4 - Initial LLDP Exchange Delay Issue**

In order to overcome this problem, the following modification is proposed.
The interval for the LLDP transmission time to refresh the timer (msgTxInterval) is set to one second for the first five transmissions after LLDP initialization and then reset to the administratively configured value. The txDelay (minimum delay between successive transmitted LLDP frames) is also set to one second as a DCBX default. This ensures that at start up both devices receive peer parameters within a short timeframe as shown in Figure 5.

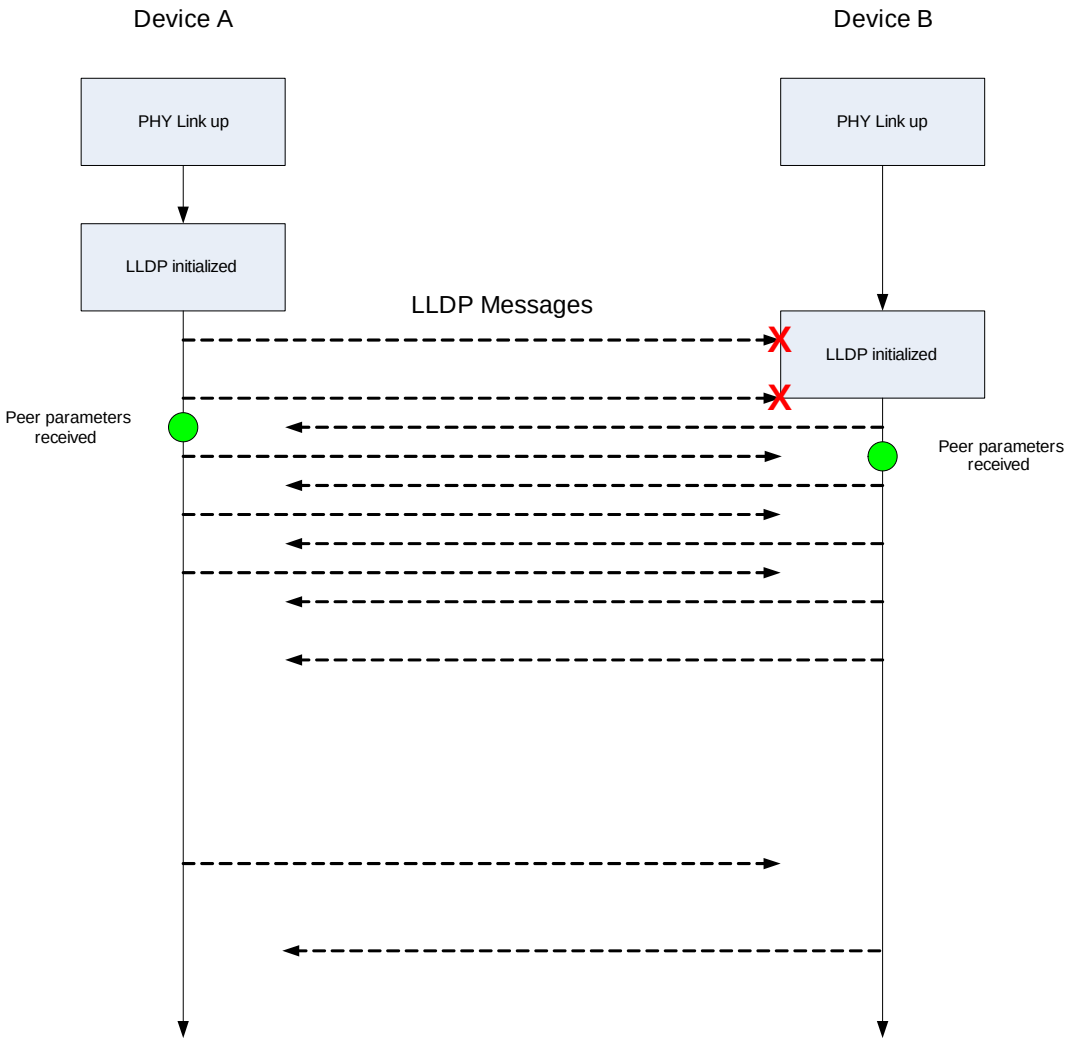


**Figure 5 - Initial Fast Retransmission of LLDP Frames**

Each time LLDP is initialized, such as link up, LLDP enters this fast transmission mode. LLDP operates in its normal transmission mode at all other times.

## 2.4 DCBX Operation

DCBX is defined as a DCBX control state machine and a set of DCB feature state machines. The DCBX control state machine handles ensuring that the two DCBX peers get in sync by exchanging LLDPDUs after link up or following a configuration change. The DCB feature state machines handle the local operational configuration for each feature by comparing and synchronizing with the peer's feature settings.

### 2.4.1 DCBX TLV Format

Information about the DCBX control state and DCB feature configuration are exchanged with the peer in DCBX TLVs that are transmitted via LLDP PDUs. Figure 6 shows the general structure of the organizationally specific DCBX TLV. The details of each sub-TLV are covered in the remainder of the document.

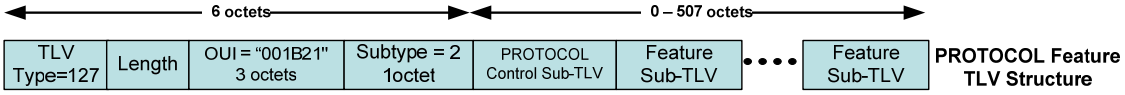


**Figure 6 - High Level DCBX TLV Structures**

The DCBX Control Sub-TLV and the set of Feature Sub-TLVs can be arranged in any order within the DCBX TLV. Duplicate Sub-TLV's (such as more than one Sub-TLV for the same feature) are not allowed. Duplicates are handled as a configuration error for the feature. A duplicate DCBX Control TLV causes an error for all features.
The DCBX sub-TLVs follow the same format as an LLDP TLV – having type, length and information fields. The type field is meaningful within the context of a DCBX TLV and the length specifies the number of octets in the information portion of the sub-TLV.

Figure 6 shows OUI=001B21 that is offered by Intel Corp. for use by all the parties using pre-standard version. Once IEEE 802.1 defines the protocol, appropriate OUI needs to be used as defined by IEEE standard.

#### 2.4.1.1 Bit and Octet Ordering Conventions

DCBX uses the same bit and octet ordering conventions as LLDP.

> [The DCBX TLV] contain[s] an integral number of octets. The octets in [a DCBX TLV] are numbered starting from one and increasing in the order they are put into the LLDP frame. The bits are numbered from zero to seven, where zero is the low-order bit.
>
> When consecutive bits within an octet are used to represent a binary number, the highest bit number has the most significant value. When consecutive octets are used to represent a binary number, the lower octet number has the most significant value. All TLVs respect these bit and octet ordering conventions, thus allowing communications to take place.

In the details that follow, the following data types are used to define structures which describe the elements of DCBX sub-TLVs:

- u32 - unsigned 32 bit integer

- u16 - unsigned 16 bit integer
- u8 - unsigned 8 bit integer

Elements listed first in a structure have lower octet numbers then subsequent elements. Bit fields within an element occupy the highest to lowest order bits of the element in the order they are listed. The following structure shows an example.

```
struct example_tlv {
    u16 type         :7;     // high order bit field in u16 element
    u16 length       :9;     // low order bit field in u16 element
    u32 fieldA       :8;     // highest order bit field in u32 element
    u32 fieldB       :8;
    u32 fieldC       :3;
    u32 fieldD       :13;    // lowest order bit field in u32 element
};

SIZE = 6 octets
```

## 2.4.2 DCBX Control State Machine

The DCBX Control state machine uses the DCBX Control sub-TLV to exchange information with the peer. In addition, it maintains some additional local state variables to manage the state machine operation. The TLV and state variables are defined in the sections that follow.

### 2.4.2.1 DCBX Control TLV

Figure 7 shows the DCBX Control TLV.

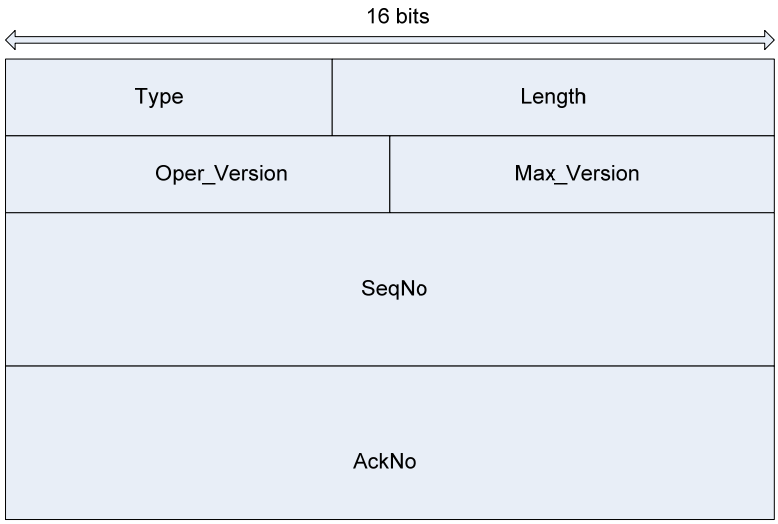


**Figure 7 - DCBX Control TLV Definition**

The following table lists the fields in the DCBX Control TLV.

**Table 1 DCBX Control TLV Fields**

| Field | Field-Type | Range | Description |
|---|---|---|---|
| Feature-Type | Integer | N/A | Type code of the DCBX Control TLV. |
| Length | Integer | N/A | Length of the DCBX Control sub-TLV payload (not including the Type and Length fields). The length is less than the maximum possible value (511) as this TLV is packaged inside the DCBX TLV along with other feature TLV's. |
| Oper Version | Integer | 0..255 | Operating version of the DCBX protocol. The system adjusts as needed to operate at the highest version supported by both link partners. |
| Max Version | Integer | 0..255 | Highest DCBX protocol version supported by the system. Version numbers start at zero. The DCBX protocol must be backward compatible with all previous versions. |
| SeqNo | Integer | 0 .. ($2^{32}$ –1) | A value that changes each time an exchanged parameter in one or more of the DCB feature TLV's changes. |
| AckNo | Integer | 0 .. ($2^{32}$ –1) | The SeqNo value from the most recent peer DCBX TLV that has been handled. This acknowledges to the peer that a specific SeqNo has been received. |

## 2.4.2.2 DCBX Control State Variables

The following table lists the local state variables used to maintain the DCBX Control state machine.

**Table 2 DCBX Control state variables**

| State Variable | Type | Range | Description |
|---|---|---|---|
| RcvdAckNo | Integer | 0 .. ($2^{32}$ –1) | The 'AckNo' from the most recent peer DCBX TLV that has been handled. This is an acknowledgement from the peer that a specific SeqNo |

| | | | |
|---|---|---|---|
| | | | has been received. |
| NoDCBXTLV Received | Boolean | TRUE/FALSE | This flag is set when somethingChangedRemote event is received from LLDP and Remote MIB indicates empty DCB TLVs |
| DCBXFeatureUpdate | Boolean | TRUE/FALSE | Indicates any change in DCBX Feature |

### 2.4.2.3 DCBX Control State Machine

In addition to the TLV fields and state variables previously described, the DCBX Control state machine uses the following mechanisms:

- **somethingChangedLocal** – this is an indication from the DCBX state machine to the LLDP module that there is a new DCBX TLV to transmit.
- **somethingChangedRemote** – this is an indication from the LLDP module to the DCBX Control state machine that there is new information from the peer (such as new DCBX TLV or data has expired).
- **DCBXFeatureUpdate** – the DCBX Control state machine provides this indication to the DCB Feature state machines after it has received the somethingChangedRemote indication.
- The **SeqNo, DCBXFeatureUpdate** and **RcvdAckNo** variables are visible to the DCB Feature state machines.

Figure 8 shows the operation of the DCBX Control state machine. Note that the diagram is defined using an infinite loop model (such as no waiting). Implementations might use event waiting mechanisms as long as the function of the state machine is preserved.

A few notes concerning the notations used in Figure 8:

- DCBX Control TLV fields and state variables are used directly such as SeqNo)
- Variables from the Feature state machines are identified by pre-pending Feature to the variable: For example, Feature.Syncd refers to a variable called Syncd from a Feature state machine.
- TLV fields received from the peer are identified as: Rx.< variable> - i.e. ‘Rx.SeqNo.

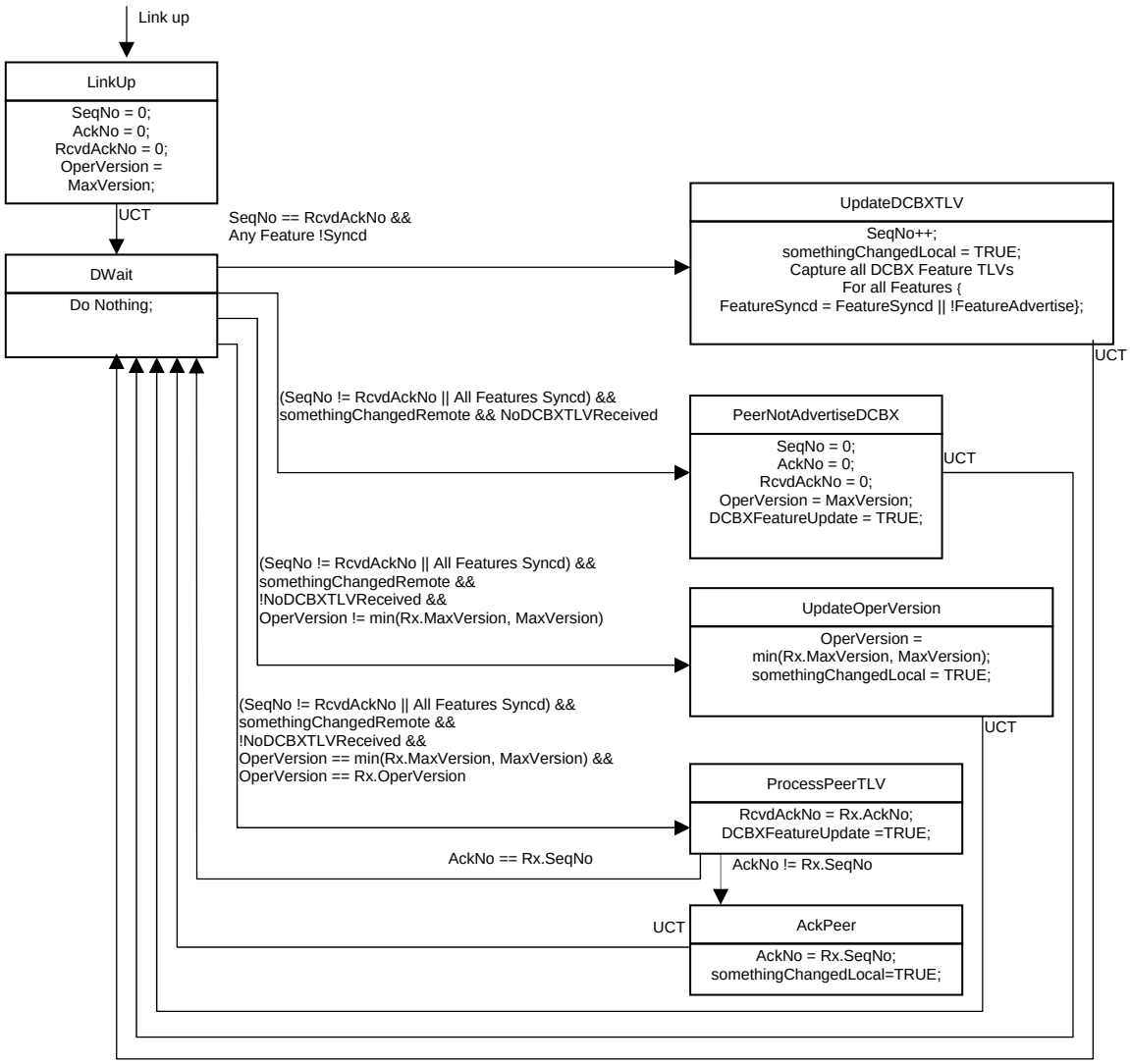


**Figure 8 - DCBX Control State Machine Diagram**

Commentary on the DCBX Control state machine diagram by reference label:

- UpdateDCBXTLV: "Capture TLV" term implies inclusion of feature TLV to be indicated to LLDP for transmission, if the feature is advertised.
- PeerNotAdvertiseDCBX If the DCBX TLV from the peer has expired, then the local side resets similar to a link up. This is a different case than an actual link down, which would cause this state machine to exit.
- AckPeer: The peer has sent a Control TLV with a new sequence number. Send a new Control TLV with an updated AckNo field.

## 2.4.3 DCB Feature State Machine

This section defines the operation of the DCB Feature state machine. The configuration of each DCB feature is managed by that feature's DCB Feature state machine. All DCB Feature state machines operate in the same manner.

### 2.4.3.1 DCB Feature TLV

Figure 9 shows the common TLV header structure used for all DCB feature sub-TLV's. Feature specific TLV parameters are pre-pended with this header.

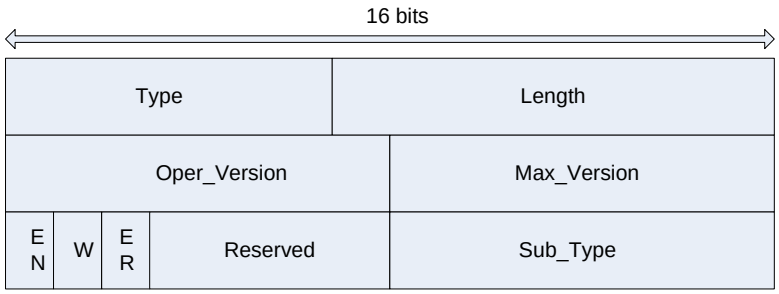


**Figure 9 – DCB Feature TLV Header Definition (DCBX_tlv_header)**

Figure 10 shows a generic DCB Feature TLV structure. The details of each feature specific parameter structure are defined in the upcoming DCB Feature sections.

```
struct PROTOCOL_feature_tlv {
    struct DCBX_tlv_header h;
    struct DCBX_feature_cfg desired_cfg;
};

SIZE = 6 + sizeof(struct DCBX_feature_cfg)
```

**Figure 10 – Generic DCB Feature TLV**

The following table lists the fields in the DCB Feature TLVs that are used to define the operation of the DCB Feature state machine.

**Table 3 DCB Feature TLV header field definitions**

| Field | Type | Range | Description |
|---|---|---|---|
| Type | Integer | 2..127 | Type code of the DCB Feature. Following is a list of defined types:<br>1 – DCBX Control (not a feature)<br>2 – Priority Groups |

| | | | |
|---|---|---|---|
| | | | 3 – Priority-based Flow Control<br>4 – Application Protocol |
| Length | Integer | N/A | Length of the DCB Feature sub-TLV payload (not including the Type and Length fields). The length is less than the maximum possible value (511) as this TLV is packaged inside the DCBX TLV along with other feature TLV's. |
| Oper Version | Integer | 0 .. 255 | Operating version of the feature. The system adjusts to operate at the highest version supported by both link partners. |
| Max Version | Integer | 0 .. 255 | Highest feature version supported by the system. Version numbers start at zero. The feature must be backward compatible for all previous versions. |
| Enable | Boolean | Truth value | Locally administered parameter that indicates whether the DCB feature is enabled or not. |
| Willing | Boolean | Truth value | Locally administered parameter that indicates whether this feature accepts its configuration from the peer or not. When set to TRUE, the system uses the DesiredCfg supplied by a !Willing peer as the OperCfg. A system set to Willing must be capable of accepting any valid DesiredCfg for the feature from the peer. If both local and remote systems have the same value for the Willing flag, then the local DesiredCfg is used and the operational outcome of the exchange is determined by the Compatible method of the feature. |
| Error | Boolean | Truth value | Indicates that an error has occurred during the configuration exchange with the peer. Error is also set to TRUE when the Compatible method for the feature fails. |

| | | | |
|---|---|---|---|
| | | | The Feature turns OperMode to FALSE if either the local or remote Error flag is set to TRUE. Duplicate TLV's for the same Type/SubType or the DCBX Control TLV also causes Error to be set to TRUE.<br><br>System errors are not reflected on this Error Flag. |
| SubType | Integer | 0..255 | Some Feature TLVs may define subtypes that are specific to that feature. When subtypes are not defined by a specific feature, subtype field should be set to zero<br><br>In general, the Type and SubType, taken together, identify a unique feature that is managed by an instance of the DCB Feature State Machine. |
| | | | |

**NOTE**: A node does not have to support 8 classes of service in order to be considered capable of accepting any valid DesiredCfg. It may fulfill the requirements of the configuration by combining priorities or priority groups requiring similar service (e.g. PFC configuration and bandwidth management) into a traffic class. Details about decision to combine various priorities in single traffic class is out of scope of this document.

**NOTE**: The TLV always carries the DesiredCfg. A system uses its own DesiredCfg, the peer's DesiredCfg (PeerCfg) and the other bits (willing, error, etc.) to derive its OperCfg. When both sides advertise, they should be able to derive the same OperCfg.

### 2.4.3.2 DCB Feature State Variables

The following table lists the additional state variables used to maintain each DCB Feature state machine.

**Table 4 DCBX state variable definitions for the DCB Feature state machine**

| State Variable | Type | Range | Description |
|---|---|---|---|
| Advertise | Boolean | Truth value | Locally administered parameter that indicates whether this feature is exchanged in the DCBX TLV. When Advertise is False, received TLVs for this feature are ignored. |
| OperMode | Boolean | Truth value | Operational state of the feature. |
| FeatureSeqNo | Integer | 0 .. | When Syncd is False, this |

| | | | |
|---|---|---|---|
| | | ($2^{32}$ –1) | indicates the value that SeqNo must become equal to before Syncd can become True. |
| Syncd | Boolean | Truth value | Indicates whether the current DesiredCfg has been received by the peer. |
| OperCfg | Structure | NA | The actual operating configuration of the feature. Derived from either DesiredCfg or PeerCfg. |
| PeerCfg | Structure | NA | The DesiredCfg of the peer – as received in a DCBX TLV from the peer. |
| DesiredCfg | Structure | NA | This represents the locally configured values of the feature specific configuration. |
| PeerWilling | Boolean | Truth value | The Willing state of the peer – as received in a DCBX TLV from the peer. |
| LocalParameterChange | Boolean | Truth Value | Indicates that a configurable DCB Feature TLV field or state variable has been modified on the local system. |

### 2.4.3.3 DCB Feature State Machine

In addition to the TLV fields and state variables previously described, the DCB Feature state machine uses the following mechanisms:

- **DCBXFeatureUpdate** – this represents an indication from the DCBX Control state machine that there is new information from the peer (such as a new DCBX TLV or data has expired).
- The **Syncd** variable from each DCB Feature state machine is visible to the DCBX Control state machine.
- Each feature has a method called **Compatible** which is used to compare the DesiredCfg and PeerCfg.
- Each feature retrieves **LocalParams** from non-volatile storage or sets them to default values at link up.
- **LocalParameterChange** indicates that a configurable DCB Feature TLV field or state variable has been modified on the local system.
- The **SeqNo,** DCBXFeatureUpdate and **RcvdAckNo** are control state machine variables that are visible to the DCB Feature state machines.
- Each feature has a method called **SetCfg** which accepts two parameters. This method is called to set operational parameters for the given feature. First parameter to this method provides configuration parameters to be used by the method. Second parameter is a Boolean.

Figure 11 shows the operation of the DCB Feature state machine. Note that the figure is defined using an infinite loop model (such as no waiting). Implementations might use event waiting mechanisms as long as the function of the state machine is preserved.

A few notes concerning the notations used in Figure 11:

- TLV fields and state variables are used directly (e.g. SeqNo)
- TLV fields received from the peer are identified as: Rx.<variable> - such as Rx.SeqNo.

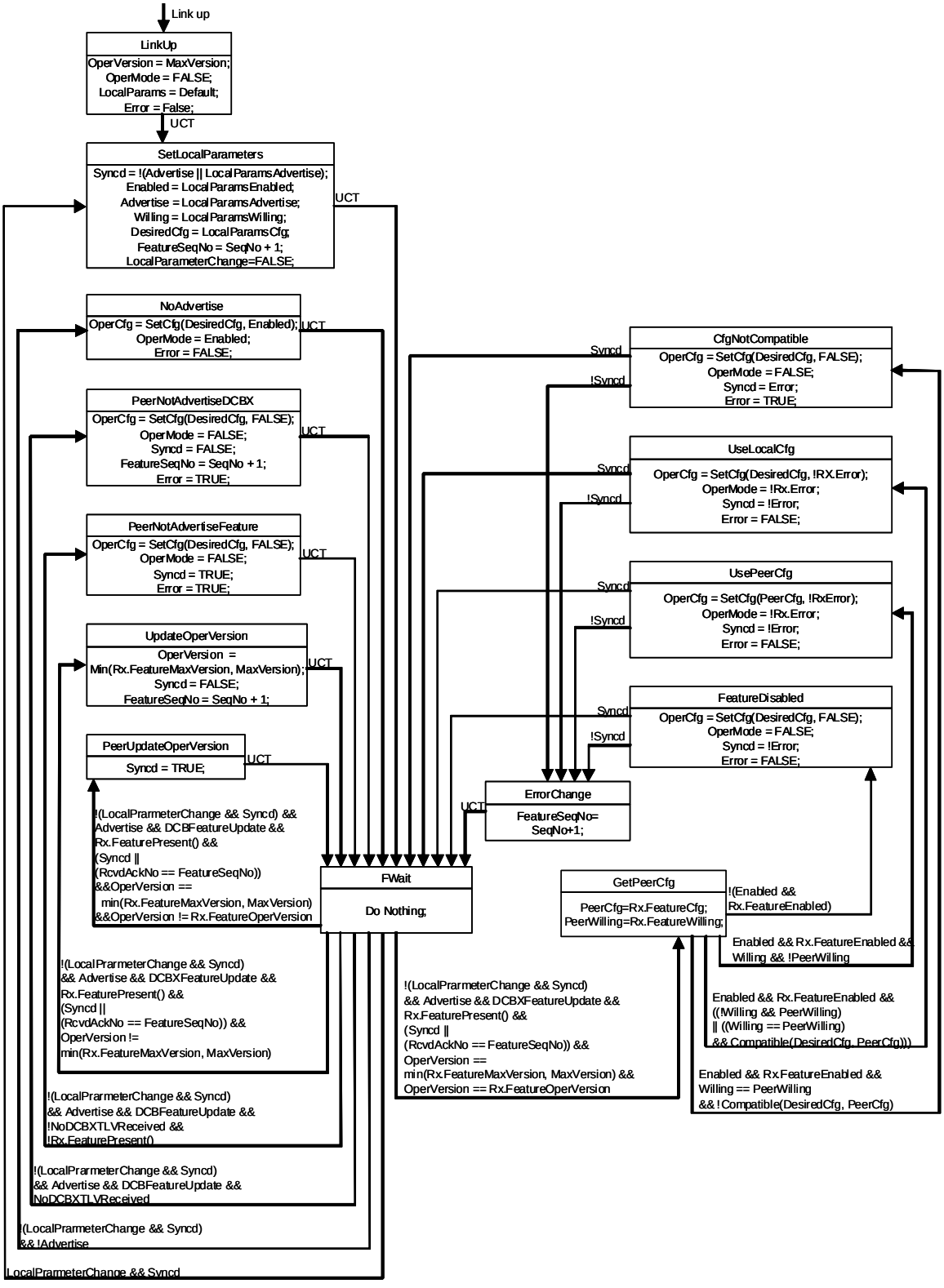


**Figure 11 - DCB Feature State Machine**

Commentary on the DCB Feature state machine by reference label:

- SetLocalParameters: If multiple features experience a change and set Syncd to FALSE, it is possible that the first change triggers the Control state machine to send an LLDP message. Additional pending changes do not get sent until the first change has been acknowledged (per D2 of Control state machine). In other words, the SeqNo's ratchet up and are acknowledged one value at a time. For example, the AckNo from the peer could be 9, the local SeqNo is 10, and multiple features could be pending with FeatureSeqNo at 11. A PDU with SeqNo 11 is not sent until an AckNo of 10 is received.
- NoAdvertise (any place OperCfg is set) – The hardware configuration for the feature takes place at the point OperCfg is set and the OperMode is set. The implementation might keep track of whether or not the OperCfg and OperMode have actually changed and require an update to the hardware configuration.
- CfgNotCompatible – Error is explicitly set to TRUE here to indicate that the two peers have a DCB configuration that is not compatible.
- SetCfg: When second parameter (Boolean) is set to TRUE, then configuration passed by first parameter is applied to Operational values. When second parameter is "FALSE", then behavior is feature dependent and will be defined in the relevant sections for each feature.

### 2.4.4 Manager Notifications

Implementations might choose to generate notifications when certain events occur. These types of events could include:

- Conditions indicating possible configuration error –for example, when the Compatible method fails.
- Conditions where the feature is not present on the peer. This can happen when a device does not support a feature (not really an error) or if the feature's Advertise flag is off (possible configuration error).
- Configuration received from peer results into partial or complete mismatch.
- The peer stops responding – as evidenced by an LLDP timeout event (delivered via the somethingChangedRemote indication).
- Each time the Error flag is set to TRUE.

# 3. DCB Features

This section defines the DCB Feature parameters and statistics.

## 3.1 Priority Group Feature

This section describes the details of the Priority Group feature. The Priority Groups Specification provides configuration tables as well as a scheduling algorithm for managing bandwidth for various traffic classes on a converged link.

**NOTE**: Although it is expected that DCB devices will eventually provide scheduling functionality as specified in the Priority Group specification (or better), legacy implementations exist. To encourage wider adoption, this Priority Group Feature allows legacy implementations to match scheduler capabilities to the behavior implied by the Priority Group specification as close as possible. All DCBX implementations must be capable of advertising the Priority Group TLV.

### 3.1.1 Priority Group Parameters

The following table lists the Priority Group parameters.

**Table 5 - Priority Group Parameters**

| Parameter | Syntax | Range | Scope | Description |
|---|---|---|---|---|
| NumTCsSupported | Integer | 0..7 | Exchanged | Number of TCs supported by device.<br><br>Number of Priority Groups supported by a device can not be more than number of TCs supported. |
| Priority Group (PG) Allocation | Table | | | |
| PG ID (index) | Integer | 0..15 | Exchanged | Queue bandwidth group |
| PG Percentage | Integer | 0..100 | Exchanged | Percentage of link bandwidth |
| Priority Allocation | Table | | | |
| Priority (index) | Integer | 0..7 | Exchanged | |
| PG ID | Integer | 0..15 | Exchanged | PG to which the priority belongs |

### 3.1.2 Priority Group TLV

Figure 12 shows Priority Group parameters structure that is used in the Priority Group Feature TLV.

```
struct dcbx_pg_cfg {
      u8 pgid_0  :4;          /* PGID of priority 0 */
      u8 pgid_1  :4;          /* PGID of priority 1 */
      u8 pgid_2  :4;          /* PGID of priority 2 */
      u8 pgid_3  :4;          /* PGID of priority 3 */
      u8 pgid_4  :4;          /* PGID of priority 4 */
      u8 pgid_5  :4;          /* PGID of priority 5 */
      u8 pgid_6  :4;          /* PGID of priority 6 */
      u8 pgid_7  :4;          /* PGID of priority 7 */
      u8 pg_percentage[8];  /* Index is PGID */
      u8 num_tcs_supported;
}
SIZE = 13 octets
```

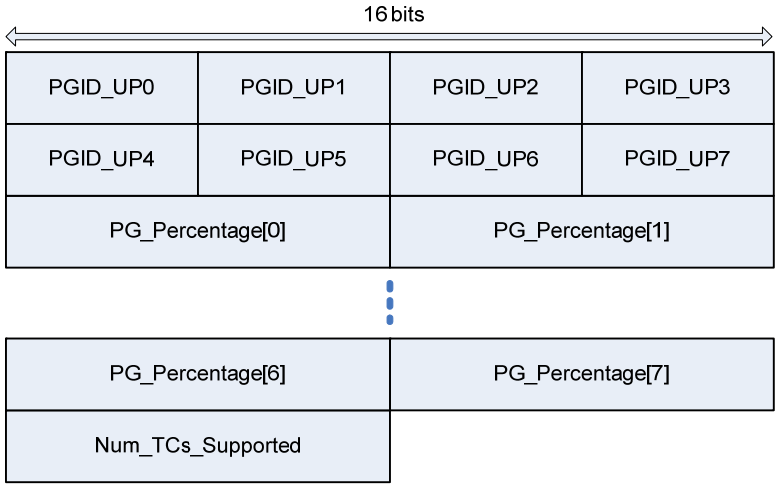


**Figure 12 - Priority Group Parameters Structure**

## 3.1.3 Priority Group Parameter Comparison

Table 6 lists how the Priority Group parameters of the local and peer nodes are compared to determine if they match or not.

**Table 6 - Priority Groups Parameter Comparison**

| Parameter | Comparison |
|---|---|
| Priority Group (PG) Allocation | |
| PG ID (index) | Does not need to match |
| PG Percentage | Does not need to match |
| Priority Allocation | |
| Priority (index) | Does not need to match |
| PG ID | Does not need to match |
| Number of TCs Supported | Does not need to match |

## 3.1.4 Feature Specific Behavior for Priority Group TLV

Priority group has specific behavior defined for SetCfg method in feature state machine. Based on second parameter to SetCfg function, following actions shall be taken for Priority Groups feature:

- When Boolean is set to TRUE, configuration passed by first parameter to the method shall be applied to operational values. Feature shall be “Enabled”.
- When second parameter (Boolean) is set to FALSE, local configuration (DesiredCfg) shall be applied to operational values and feature shall be “Enabled”.

## 3.2 Priority-based Flow Control (PFC) Feature

This section describes the details of the Priority-based Flow Control feature. This feature is important to provide “no-drop” packet delivery for certain traffic classes while maintaining existing LAN behavior for other traffic classes on converged link.

NOTE: Legacy implementations that do not support Priority-based Flow Control can signal this by setting "Enable" to FALSE. This effectively disables the Priority-based Flow Control feature at which time the peers fall back to configured 802.3X PAUSE behavior. All DCBX implementations must be capable of advertising the Priority-based Flow Control TLV.

### 3.2.1 Priority-based Flow Control Parameters

Table 7 lists the Priority-based Flow Control parameters.

**Table 7 - Priority-based Flow Control**

| Parameter | Syntax | Range | Scope | Description |
|---|---|---|---|---|
| NumTCPFCSupported | Integer | 1..8 | Exchanged | Number of TCs that can simultaneously support PFC. |
| PFC Config | Table | | | |
| Priority (index) | Integer | 0..7 | Exchanged | Priority value as defined 3-bit field by 802.1Q |
| Admin mode | Integer | 0..1 | Exchanged | Administrative PFC mode.<br>0: Disabled<br>1: Enabled<br>PFC Enabled means that flow control in both directions (Rx and Tx) is enabled. |

### 3.2.2 Priority-based Flow Control TLV

Figure 13 shows the Priority-based Flow Control parameters structure that is used in the Priority-based Flow Control Feature TLV.

```
struct dcbx_pfc_cfg {
        u8 pfc_enable;  /* bitmap of priorities with PFC enabled */
        u8 num_tcpfcs_supported;
}
```

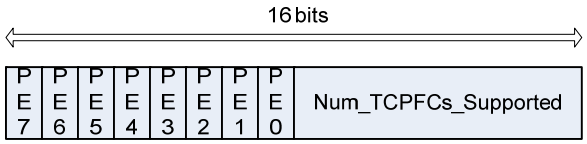


**Figure 13 - Priority-based Flow Control Parameters Structure**

### 3.2.3 Priority-based Flow Control Parameter Comparison

Local and remote parameter comparison for Admin Mode is done as follows:

```
foreach (user_priority)
{
    if ((localAdminMode == Disabled == remoteAdminMode)
            ||
        (localAdminMode == Enabled == remoteAdminMode))
    {
        Comparison successful – configuration match …
    }
    else
    {
        Comparison fails – configuration mismatch …
        break
    }
}
```

### 3.2.4 Feature Specific behavior for PFC TLV

PFC TLV has specific behavior defined for SetCfg method in feature state machine. Based on second parameter to SetCfg function, following actions shall be taken for Priority Groups feature:

- When Boolean is set to TRUE, configuration passed by first parameter to the method shall be applied to operational values. Feature shall be “Enabled”.
- When second parameter (Boolean) is set to FALSE then feature shall be “Disabled”.

## 3.3 Application Protocol Feature

This TLV allows DCB node to announce upper layer protocols and associated priority-map over DCB link. These ULPs may include L2 or L3 or L4 protocols. A DCB node can advertise multiple ULPs using a single Application Protocol TLV. Each upper level protocol is specified using a fixed size structure as shown in Figure 15 - Application Protocol Parameters Structure. The Length field of the Application Protocol TLV will include all the application protocol parameter structures specified in the TLV. For N application protocols specified in the TLV, the Length value will be sum of lengths of all the application protocol Parameters (as defined in Table 8) and the remaining Feature TLV header (except Type and Length field as defined in Table 3).

There are multiple ways to specify the upper layer application protocol. One way recommended here is to use OUI specified in DCBX Base Specification 0x001B21 (Donated by Intel Corp. for use in DCBX protocol) and carry an Application Protocol ID.

This protocol ID could be EtherType (for L2 protocols) or TCP/UDP socket number for L4 protocols.

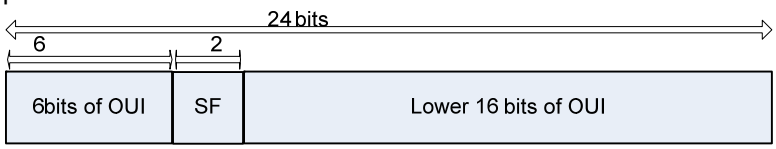


**Figure 14 - DCBX Specified use of OUI**

This proposal defines use of two least significant bits of MSB (named as SF: Selector Field) follows:
$00_2$: Application Protocol ID is L2 EtherType
$01_2$: Application Protocol ID is Socket Number (TCP/UDP)
$10_2$: Reserved
$11_2$: Reserved

**NOTE**:
E.g. FCoE protocol will be defined as follows:
SF = $00_2$, Application Protocol Id = 0x8906 or
SF = $00_2$, Application Protocol Id = 0x8914
E.g. iSCSI protocol will be announced as follows:
SF = $01_2$, Application Protocol Id = 3260

### 3.3.1 Application Protocol Parameters

**Table 8 – Application Protocol Parameters**

| Parameter | Syntax | Range | Description |
|---|---|---|---|
| **App Protocol Config** | **Table** | | |
| App Protocol Index (index 1) | Integer | 0..MAX_DCBX_APP_PROTOCOL | Application protocol Index. MAX_DCBX_APP_PROTOCOL is defined to be 15. |
| Priority (index 2) | Integer | 0..7 | Priority value as defined 3-bit field by 802.1Q. |
| SF | Integer | 0..3 | 0x00: App Proto ID carries L2 EtherType<br>0x01: App Proto ID carries Socket Number (TCP/UDP)<br>0x10: Reserved<br>0x11: Reserved |
| OUI | Integer | 24 Bit OUI Value | DCBX Protocol uses value 0x001B21 |
| Application Protocol Id | Integer | 0..65535 | This two-byte field identifies protocol supported by DCB node |
| Status | Integer | 0..1 | Status of this entry<br>0: Disabled |

| | | | |
|---|---|---|---|
| | | | 1: Enabled |

This Table 8 provides configuration for multiple application protocols and each application protocol configuration with multiple priorities.

The table parameters are translated to the Application Protocol TLV as shown in Figure 15. An application protocol using multiple priorities will use a single Application Protocol Parameter entry in the TLV with appropriate bits set in "user_priority_map" field.

## 3.3.2 Application Protocol TLV

Figure 15 shows the Application Protocol parameter structure that is used in the FCoE Application Feature TLV.

```
struct DCBXapplication_protocol {
    u16 application_protocol_id;
    u8  upper_6_bits_OUI:6; /* OUI = 0x001B21 */
    u8  selector_field: 2;
    u16 lower_16_bits_OUI;  /* OUI = 0x001B21*/
    u8  user_priority_map :8;
};
```

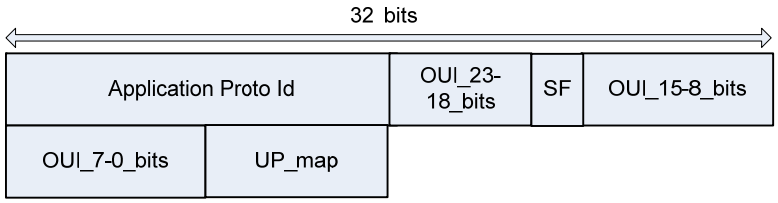


**Figure 15 - Application Protocol Parameters Structure**

A node can announce support for multiple protocols in single Application Protocol TLV using one application protocol parameter structure for each protocol advertized.

Following example shows Application Protocol TLV carrying two protocol declarations:

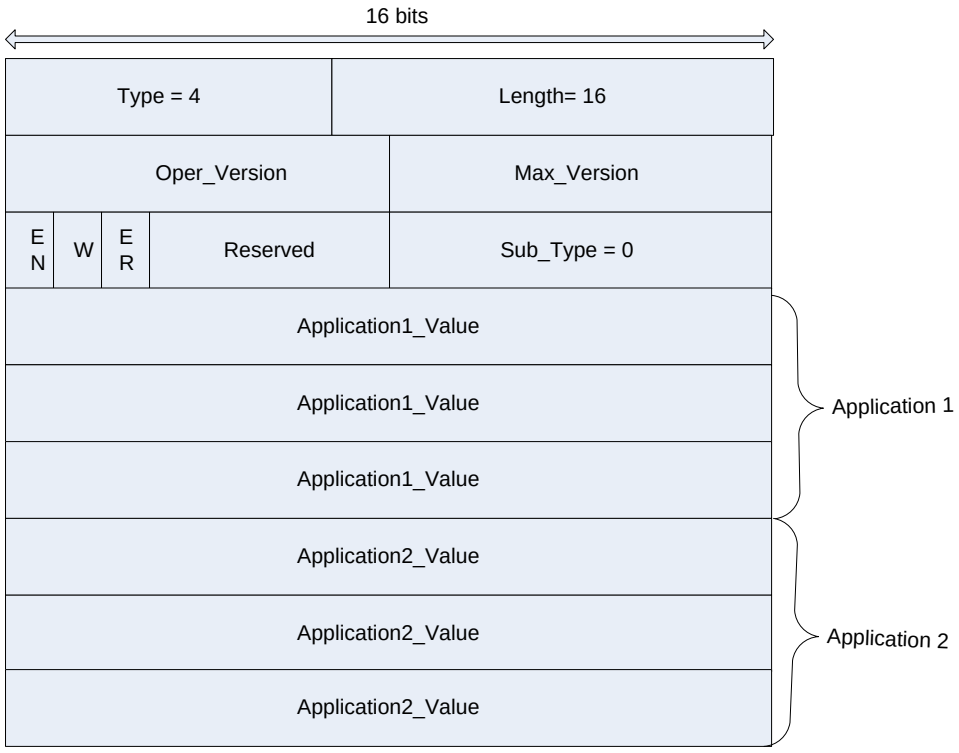


**Figure 16 Example Application Protocol TLV with Multiple Protocols**

As stated earlier, multiple Application Protocol Parameters may be contained in a single Application Protocol TLV. The “Enable”, “Willing” and “Error” bits apply to the Application Protocol TLV as a whole.

These bits have the following semantics:
- The "Enable" setting applies to all of the Application Protocol Parameters.
- The "Willing" setting applies to all of the Application Protocol Parameters.

## 3.3.3 Application Protocol Parameter Comparison

For any protocol requiring bi-directional communication in the priority there needs to be intersection between “priority map” declared in the Application Protocol Parameter.

When both peers have the same “Willing” value, the "Error" bit is set when there is a bitmap field mismatch in one or more Application Protocol Parameters common to both peers.

When a node receives peer announcement of an Application Protocol that is not supported by it, a management notification should be issued (regardless of the “Willing” settings of both peers). However “Error” bit is not set in this case.

Mismatch is expected to be notified to management entities on both sides. Further action on mismatch is beyond scope for this document.