IEEE 802.1Q YANG Bridge Port Interface Model in Support of 802.1AX, 802.1X, etc.

Marc Holness
Version 0.4
28 July 2016
IEEE 802.1Q Bridge Port

- Each Bridge Port is associated with one Interface, and in most situations, each Bridge Port is associated with a different interface
- However, there are situations in which multiple Bridge Ports are associated with the same interface
  - For example, several Bridge Ports can each correspond one-to-one with several Ethernet private lines (or SDH virtual circuits) but all on the same Interface
  - Or multiple Bridge Ports can each correspond to a single internal LAN (I-LAN) port
- Alternatively, there is the Link Aggregation (IEEE Std 802.1AX) case where there are many physical Ports for one Bridge Port
IEEE 802.1Q Bridge Port

- The MAC Relay Entity handles the media access method-independent functions of relaying frames among Bridge Ports. It uses the EISS (6.8, 6.9) provided by each Bridge Port.

- Each Bridge Port also functions as an end station and shall provide the MAC Service to an LLC Entity that operates LLC Type 1 procedures to support protocol identification, multiplexing, and demultiplexing, for PDU transmission and reception by the Spanning Tree Protocol Entity and other higher layer entities.
CFM Maintenance Point Placement

- CFM entities are specified as shims that make use of and provide the ISS or EISSS at SAPs within the network.
- The relationships among MPs, and between the MPs and the other entities in a Bridge, are configurable.
Bridge Port Interface Stack Models
Interface Stack Diagram Representation

- A SAP is an abstraction and does not necessarily correspond to any concrete realization within a system
- The entities that support a particular SAP compose an interface stack
- Each YANG Interface definition contains an interface stack table

**Object**

**Interface Reference**
- Pointer or Index

**Interface Service Data**

**Interface**
- Service Access Point (SAP)

**Shim/Service-1**
- Interface Type 1

**Shim/Service-2**
- Interface Type 2

**Interface Index (1)**
- Interface Type: Service-1
- Interface Augments: (Service-1)
- Higher Layer: ()
- Lower Layer: (2)

**Interface Index (2)**
- Interface Type: Service-2
- Interface Augments: (Service-2)
- Higher Layer: (1)
- Lower Layer: ()
Basic Bridge

- Bridge Ports are associated with Bridge components
Basic Bridge (Port) Models

Model-1
- Bridge Ports are assigned to an Interface that is independent of the underlying MAC (or service)
- Bridge Port data and MAC data attributes are associated with separate Interfaces

Model-2
- Bridge Ports are underlying MAC (or service) share the same Interface
- Bridge Port and MAC specific data attributes associated with same Interface
Basic Bridge with Link Aggregation

- “Agg Port” specific configuration gets applied
- “Aggregator” specific configuration gets applied
- System then utilizes LACP signaling to tie together aggregation members with the Aggregator
Evolution of Bridge Port Model-1 — Link Aggregation

**NOTE:** This is the preferred way to model the Aggregator. That is, simply augment the Bridge Ports versus introducing additional Aggregator specific Interfaces.
Evolution of Bridge Port Model-1
— Link Aggregation

• The Bridge Port Interfaces (A) and (B) are extended to include AGGREGATOR specific configuration and operational data

• Interface (Y) and (Z) will be extended to include AGGREGATION PORT specific configuration and operational data
  – From a YANG perspective, this is an augmentation

• LACP operation determines which AGGREGATION PORT points to which AGGREGATOR
Evolution of Bridge Port Model-1
— Link Aggregation

**NOTE:** This is not the preferred way to model the Aggregator. Augmenting the Bridge Ports versus introducing additional Aggregator specific Interfaces is more optimal.
Evolution of Bridge Port Model-1
— Link Aggregation

• Interfaces (C) and (D) are created and extended to include AGGREGATOR specific configuration and operational data
  – The Bridge Port Interfaces (A) and (B) are connected to the respective AGGREGATOR Interfaces

• Interface (Y) and (Z) will be extended to include AGGREGATION PORT specific configuration and operational data
  – From a YANG perspective, this is an augmentation

• LACP operation determines which AGGREGATION PORT points to which AGGREGATOR
Evolution of Bridge Port Model-2
— Link Aggregation
Evolution of Bridge Port Model-2
— Link Aggregation

• The original Interfaces (Y) and (Z) need to be destroyed
  – New Interfaces (W) and (X) need to be created
  – New Interfaces (A) and (B) need to be created

• Bridge Port configuration and operational data (associated with Interface (Y) and (Z)) need to be seamlessly transferred to Interfaces (A) and (B)
  – Interface (A) and (B) will be extended to include AGGREGATOR specific configuration and operational data

• Ethernet configuration and operational data (associated with Interface (Y) and (Z)) need to be seamlessly transferred to Interfaces (W) and (X)
  – Interface (W) and (X) will be extended to include AGGREGATION PORT specific configuration and operational data

• Conclusion: This model does not gracefully handle the dynamism of Link Aggregation (802.1AX)
Evolution of Bridge Port Model-1 — MAC Security and LAG

NOTE: Is this is problem? The AGGREGATION PORTS are still associated with Interface (Y) and (Z), instead of Interface (M) and (N) which are connected to the AGGREGATORS.
Evolution of Bridge Port Model-1 — MAC Security and LAG

NOTE: Is think this works, but the Aggregation Port (configuration and operational) data need to move from Interface (Y) and (Z) to Interface (M) and (N). This may be problematic.
Evolution of Bridge Port Model-1
— MAC Security and LAG

- A PAE (service shim) is inserted in the Interface stack
  - The PAE/SECY/PAC is associated with Interface (M) and (N)
Evolution of Bridge Port Model-1 — MACSec and HLE
Evolution of Bridge Port Model-1
— MACSec and HLE

- Higher Layer Entity (HLE) references Bridge Port Interface (e.g., (A))

- HLE specific configuration and operational data extends Bridge Port Interface (A) data

- PAE MACSec configuration and operational data extends Bridge Port Interface (A)
Evolution of Bridge Port Model-2 — MACSec and HLE
Evolution of Bridge Port Model-2  
— MACSec and HLE

- The original Interfaces (A) need to be destroyed and recreated
  - Before the configuration and operational data associated with this Interface includes {Bridge Port, Ethernet, HLE}
  - However, after MACSec is configured, the configuration and operational data associated with this new Interface (B) includes {Bridge Port, HLE, PAE, SecY}

- Additionally, a new Interface (Z) needs to be created
  - This Interface contains the Ethernet configuration and operational data
  - The Interface Stack will get updated accordingly

- The Bridge Port pointers within the Component and HLE needs to change

• **Conclusion:** This model does not gracefully handle the dynamism of introducing MACSec where a High Layer Entity (e.g., Spanning Tree, or IP, etc.) is involved with the Bridge Port
Conclusions/Observations
Conclusions/Observations

- Model 2 does not (gracefully) support insertion of protocol entities and Interfaces within an existing Interface stack

- For example,
  - Introduction of certain protocols (e.g., 802.1AX), which are dynamic in nature, results in existing Interface tear down and data (both configuration and operational) needs to be transferred (in real-time) to newly created Interfaces that are located elsewhere in the Interface stack
  - Higher Layer Entities (e.g., Spanning Tree, IP, etc.) also need to access the Interface stack of the Bridge Port. There can be configuration and operational data associated with the HLE that needs to be resident with the Bridge Port Interface
Conclusions/Observations

• **Conclusion**: Model-1 appears to support the flexibility of the Bridge Port as specified in 802.1Q-2014
Backup Material

Bridge Port Interface
IEEE 802.1 Objects Within YANG Object Hierarchy
IETF Interface Management Model

- IETF Interface Management Model (RFC 7223) can be represented using UML as shown below.

```
interfaces
  string name;  // r-w
  string description;  // r-w
  if-type type;  // r-w
  bool enabled;  // r-w
  enum link-up-down-trap-enable;  // r-w

interfaces-state
  string name;  // r
  if-type type;  // r
  enum admin-status;  // r
  enum oper-status;  // r
  date-time last-change;  // r
  int32 ifIndex;  // r
  address phys-address;  // r
  if-ref higher-layer-if;  // r
  if-ref lower-layer-if;  // r
  gauge64 speed;  // r
```

Statistics

```
date-time discontinuity-time;  // r
counter64 in-octets, in-unicast-pkts, in-broadcast-pkts, in-multicast-pkts;  // r
counter64 in-discards, in-errors, in-unknown-protos;  // r
counter64 out-octets, out-unicast-pkts, out-broadcast-pkts, out-multicast-pkts;  // r
counter64 out-discards, out-errors;  // r
```
The Generic Bridge YANG Model

```yang
* bridge
  string name; // (12.4) r-w
  macAddress address; // (12.4) r-w
  enum type; // r-w
  int ports; // (12.4) r
  counter32 up-time; // (12.4) r
  int components; // r

component
  string name; // r-w
  int id; // (12.3) r-w
  enum type; // (12.3) r-w
  macAddress address; // (8.13.8, 13.24) r-w
  bool traffic-class-enabled; // (12.4.1.5.1) r-w
  bool mmp-enabled-status; // (12.4.1.5.1) r-w
  int ports; // (12.4.1.1.3) r

* bridge-port // r

struct capabilities // (12.4.1.5.2) r

filtering-database
  int aging-time; // (12.7, 8.8.3) r-w
  int size; // (12.7) r
  int static-entries; // (12.7, 8.8.1) r
  int dynamic-entries; // (12.7, 8.8.5) r
  int static-vlan-registration-entries; // (12.7, 8.8.2) r
  int dynamic-vlan-registration-entries; // (12.7, 8.8.5) r
  int mac-address-registration-entries; // (12.7, 8.8.4) r

permanent-database
  int size; // (12.7.6) r
  int static-entries; // (12.7.6) r
  int static-vlan-registration-entries; // (12.7.6) r

* bridge-vlan
  int version; // (12.10.1.3) r
  int max-vids; // (12.10.1.3) r
  bool override-default-pvid; // (12.10.1.3) r
  struct protocol-template; // (12.10.1.7) r
  int max-msti; // (12.10.1.7) r

* vids

* vid-to-fid-allocation

* protocol-group-database
  string frame-format-type; // (12.10.1.7) r-w
  struct frame-format; // (12.10.1.7) r-w
  int protocol-group-id; // (6.12.2) r-w

* vid-to-fid-allocation
  int *vids; // (12.10.1.7) r-w
  int *vid; // (12.10.3) r-w
  enum allocation-type; // (12.10.3) r-w

* port-map
  if-ref port-number; // (8.8.1) r-w
  struct map; // (8.8.1) r-w

* port-map
  if-ref port-number; // (8.8.1) r-w
  struct map; // (8.8.1) r-w

* port-map
  if-ref port-number; // (8.8.1) r-w
  struct map; // (8.8.1) r-w

```

The list of Bridge Ports

---

Marc Holness (mholness@ciena.com) - 29
# The Bridge Port Interface Model

The Bridge Port Interface Model is a YANG module that defines a bridge port interface model for network interfaces. The module includes definitions for interface names, types, and various statistics and configurations for bridge ports.

## Interfaces

The `interfaces` module defines various fields for interface configurations, including:

- `string name;`  
- `string description;`  
- `string if-type type;`  
- `bool enabled;`  
- `enum link-up-down-trap-enable;`  

## Bridge Port

The `bridge-port` module specifies the bridge port interface model, including:

- `string component-name;`  
- `if-ref service-if`  
- `int default-priority;`  
- `struct priority-regeneration-table;`  
- `bool use-dei;`  
- `bool drop-encoding;`  
- `enum service-access-priority-selection;`  
- `struct service-access-priority;`  
- `struct traffic-class-table;`  

## Bridge Port Statistics

The `bridge-port-statistics` module includes statistics for bridge ports:

- `counter64 delay-exceeded-discards, mtu-exceeded-discards;`  
- `counter64 frame-rx, octets-rx, frame-tx, octets-tx;`  
- `counter64 discard-inbound, forward-outbound, discard-lack-of-buffers;`  
- `counter64 discard-on-ingress-filtering, discard-ttl-expired;`  

## Interface Stack of Bridge Port

The diagram illustrates the relationship between the interfaces and the bridge port, showing how configurations and statistics flow through the stack.