March 15, 2010

EVB Proposal for:
Virtual Ethernet Port Aggregator,
Edge Virtual Bridging TLV,
Edge TLV Transport Protocol,
Multichannel, and
VSI Discovery Protocol

Version 0, Rev 0.1

Abstract: This document is a proposal for the development of IEEE Edge Virtual Bridging (EVB) technologies. These proposals cover a suite of mechanisms that may be used to construct an EVB-based solution including architectural overview, discovery, management objects, and state machines.

Keywords: VEPA, VEB, Multi-Channel, ETTP, VDP

This document is a proposal for Virtual Ethernet Port Aggregator, Edge Virtual Bridging TLV, Edge TLV Transport Protocol, Multichannel, and VSI Discovery Protocol.

Editors: Hewlett-Packard Corp., IBM

Contributing Authors

Company	Contacts
BNT	Daya Kamath
BNT	Jay Kidambi
BNT	Vijoy Pandey
Broadcom	Uri Elzur
Brocade	Anoop Ghanwani
Chelsio	Asgeir Eiriksson
Emulex	Chait Tumuluri
HP	Paul Bottroff
HP	Paul Congdon
HP	Chuck Hudson
HP	Michael Krause
IBM	Vivek Kashyap
IBM	Renato Recio
IBM	Rakesh Sharma
Juniper	Srikanth Kilaru
QLogic	Manoj Wadekar

TABLE OF CONTENTS

LIST OF TABLES

Error! No table of figures entries found.

LIST OF FIGURES

Related Documents

Specifications	Company

Change History

By	Date	Details

1. Document Scope

This document details Virtual Ethernet Port Aggregator (VEPA) theory of operation and the discovery and capability exchange protocol used to support a VEPA solution. VEPA relies upon LLDP to provide discovery and capability exchange. These exchanges occur between a physical end station and an adjacent bridge.

1.1 Purpose

The purpose of this proposal is to define a proposal for **Discovery and Configuration of Ethernet Virtual Bridging (EVB) capabilities, using: Multi-Channel, Edge TLV Transport Protocol (ETTP) and Virtual Station Interface (VSI) Discovery and Configuration Protocol (VDP).** These protocols are used to determine EVB, Multichannel, ETTP and VDP capability presence within a physical end station and an adjacent bridge.

2. Introduction

Evolving standards combined with the growing size of enterprise and cloud-based networking deployments has led to a significant increase in the complexity of Ethernet networking in the data center. The advent of virtualization technology has compounded this complexity due to the significant increase in the number of Ethernet switches and the change in the solution deployment scenario. Hypervisors have incorporated Virtual Ethernet Bridges (VEB) into the physical end station effectively adding one or more Ethernet switches per end node. A VEB is a frame relay service that supports local bridging between multiple virtual end stations (an internal private virtual network) and (optionally) the external bridging environment. A VEB may be implemented in software as a virtual switch (vSwitch) as illustrated in Figure 1 or as embedded hardware within a Network Interface Controller (NIC) as illustrated in Figure 2.

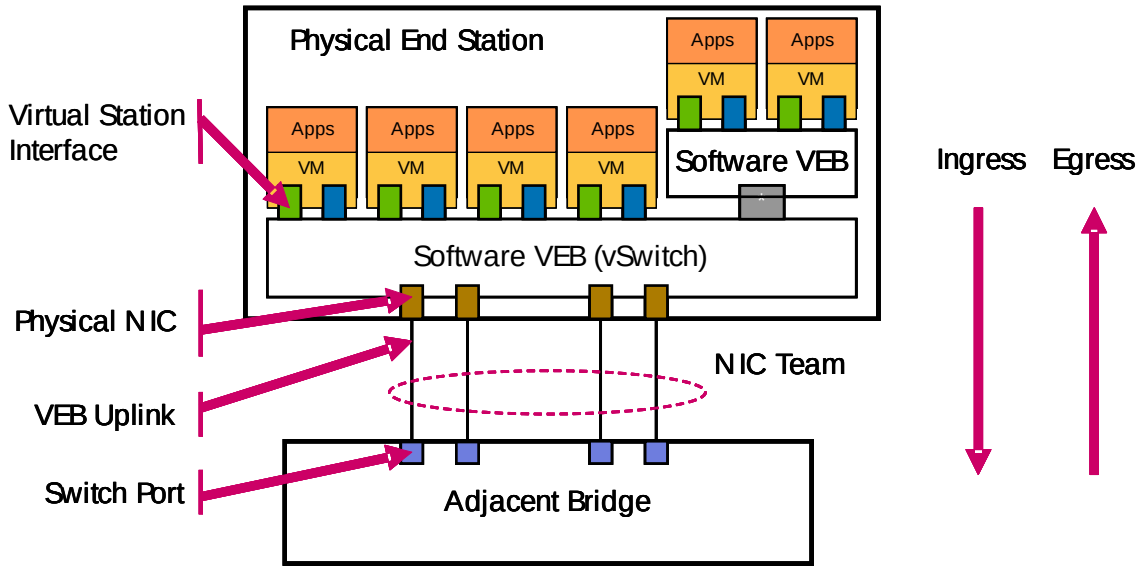


Figure 1. Example Physical End Station with Multiple VM and Two Software VEB

Figure 1 illustrates the following:

- Each VM may support one or more Virtual NICs.
  - Typically, a VM will support a virtual NIC (vNIC) that emulates a physical NIC. Each vNIC will contain a Virtual Station Interface (VSI) which is connected to a VEB.
- A VEB supports a single logical uplink to the external adjacent bridge. Multiple uplinks can be teamed via 802.3ad or other techniques.
- A software VEB (vSwitch) is typically implemented within a hypervisor requiring each VM I/O operation to trap to the hypervisor for processing.
  - Hypervisor traps consume system resources and can lead to varying performance loss depending upon the number of I/O operations per second and the amount of rich network functionality performed per operation.

- Being in the hypervisor allows a software VEB to support one or more physical NICs.
- VEB may be cascaded to provide modularity or additional fan-out.
- Not shown but important to note is a VEB does not require any modifications to the Ethernet frame to operate.

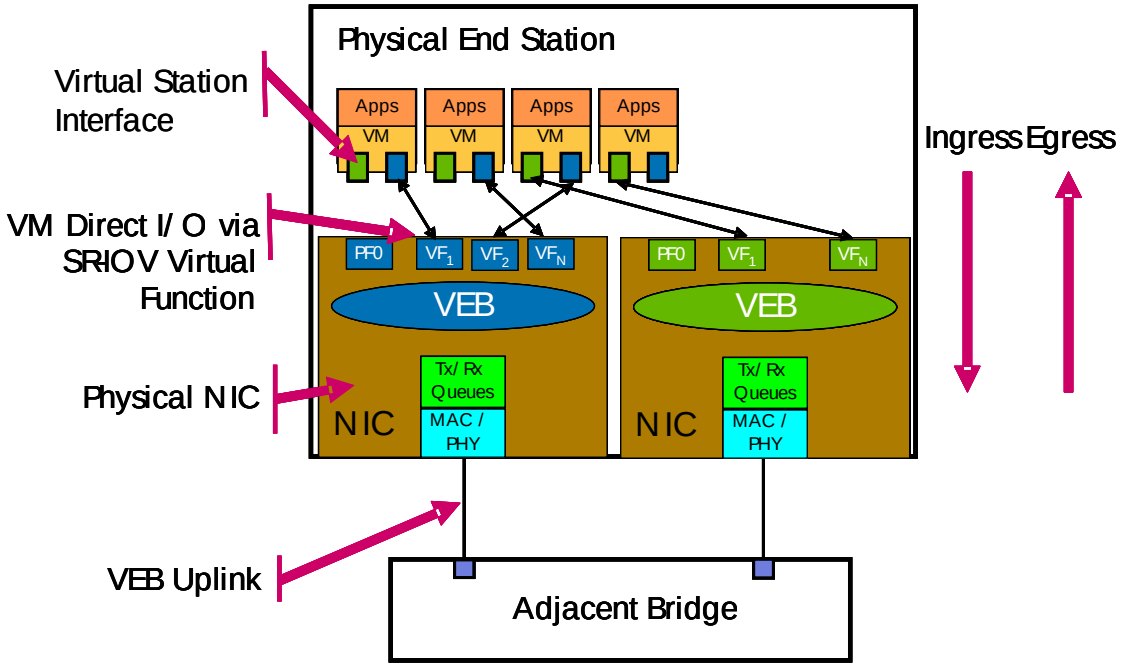


Figure 2. Example Physical End Station with Multiple Hardware VEB

Figure 2 illustrates the following:

- Each physical NIC supports
  - One (or more) physical ports attached to an adjacent bridge
    - Each physical port represents a single VEB uplink.
  - One or more hardware-embedded VEB. An embedded VEB cannot span multiple physical NIC.
  - Direct I/O support via SR-IOV Virtual Functions (VF).
    - Direct I/O allows a VM to bypass the hypervisor and directly access the NIC to send / receive packets. Bypassing the hypervisor reduces system resource consumption allowing higher performance solutions than traditional software VEB.
- Each VM may support one or more Virtual NICs.
  - In this example, each VM supports two vNIC – one per physical NIC.
  - Each vNIC contains a VSI which is associated with a SR-IOV VF to provide Direct I/O support.

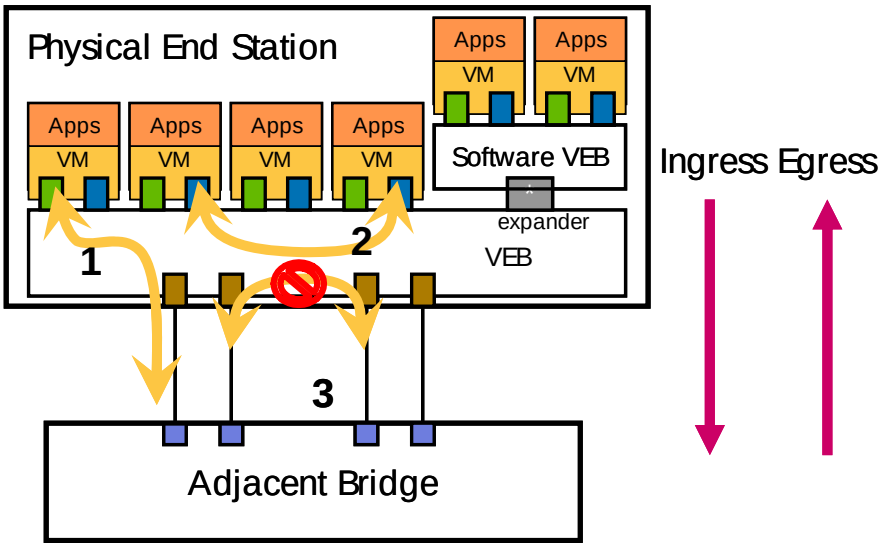


Figure 3. VEB Frame Relay Support

VEB packet forwarding supports both traditional end station-to-adjacent bridge as well as local VSI-to-VSI packet forwarding. As illustrated in Figure 3, a VEB forwards packets as follows:

- VEB forwards packets based on the MAC address and optionally via a port group or VLAN identifier.
- VEB forwards packets from a VSI to the uplink from an adjacent bridge (path 1) or between co-located VSI (path 2)
    - A NIC-embedded VEB can only forward packets between VSI attached to a common NIC. As shown in Figure 2, only VM that share the blue VEB can forward packets via the blue VEB. Similarly, only VM that share the green VEB can forward packets via the green VEB. A VM on the blue VEB cannot forward packets to a VM on the green VEB directly; a software VEB would be required to bridge the two NIC-embedded VEB.
- VEB supports only a single active logical uplink
    - Multiple uplinks can be teamed via 802.3ad or other techniques
    - Uplink-to-uplink packet forwarding is not allowed (path 3)
- VEB does not participate in or affect spanning tree operation.

VEB solutions have been shipping for a number of years and are available from multiple suppliers. Though the functional robustness of solutions will vary, local bridging via a VEB provides a number of common benefits and allows hypervisors to:

- Operate without external bridges attached
- Operate with a broad range of Ethernet environments
- Maximize local bandwidth – bandwidth is limited by end station memory and local I/O bandwidth and not by the Ethernet link bandwidth
- Minimize local latency – no incremental latency due to interaction with the external network
- Minimize packet loss, i.e. no packet loss due to external network events – external bridge or link failure, CRC error detection, congestion-based packet loss, etc.

By definition traffic between VMs connected to a VEB stay within the server. Some clients prefer the traffic to be sent through an external switch, so the external network's access

and security policies can be applied to the traffic. To address this type of requirement a a Virtual Ethernet Port Aggregator (VEPA) is documented.

A Virtual Ethernet Port Aggregator (VEPA) is a capability within a physical end-station that collaborates with an adjacent bridge to provide frame relay services between multiple co-located virtual machines (VMs) and the external network. A VEPA collaborates by:

- Forwarding all station-originated frames to the adjacent bridge for frame processing and frame relay.
- Steering and replicating frames received from the adjacent bridge to the appropriate VM destinations.
- A VEPA takes advantage of a special reflective relay forwarding mode (i.e. allow forwarding back out the port a frame was received) on the adjacent bridge to support inter-VM communication within the same physical host.
    - Clause 8.6.1 of Standard IEEE 802.1Q-2005 [11] states that when a switch reception port is in the forwarding state, each switch port in the forwarding state, *other than the reception port itself*, is a potential transmission port. A VEPA requires an exception to this rule in order to allow inter-VM traffic on the adjacent host over the single uplink. This exception distinguishes the port attached to a VEPA uplink as a VEPA-enabled port which supports forwarding in reflective relay mode.
- Similar to a VEB, a VEPA may be implemented in software or in conjunction with embedded hardware within a NIC.

The VEPA is connected to the adjacent bridge only by a single uplink connection. The connection is attached to a VEPA-enabled port on the adjacent bridge. A conceptual VEPA is shown in Figure 4.

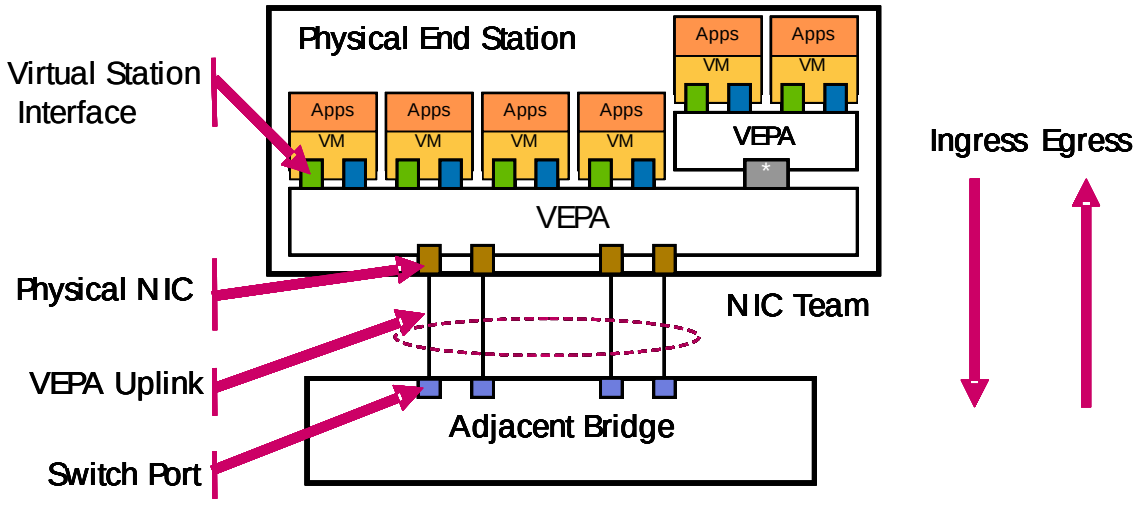


Figure 4. Example Physical End Station with Multiple VM Communicating through a VEPA

Figure 4 illustrates the following:

- Each VM may support one or more Virtual End Stations.
  - A VM supports a virtual NIC (vNIC) which emulates a physical NIC. A vNIC is attached to a VEPA via a Virtual Station Interface (VSI). A VSI is a physical or software emulated end station connected to a VEB or a VEPA.
- A VEPA supports a single logical uplink.
- Software VEPA (vSwitch) may support one or more physical NICs.
- The total number of VSI made available may be scaled by cascading VEPA in a tree as shown in Fig. 4. A VSI on a root VEPA connected to a leaf VEPA higher in the topology is known as an expander port. A root VEPA will forward all frames with an unknown destination address to the expander port. This eliminates the need for the root VEPA to comprehend all of the MAC addresses of every VM in the physical end station.
- Not shown but important to note is VEPA does not require any modifications to the Ethernet frame to operate.

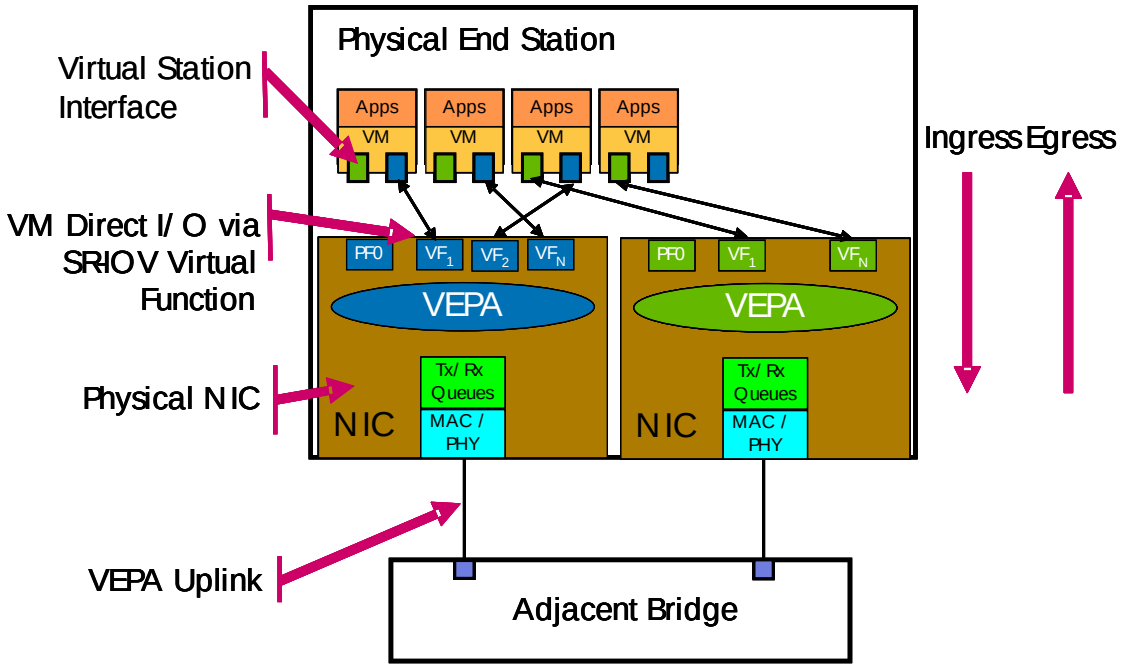


Figure 5. Example Physical End Station with Multiple Hardware VEPA

Figure 5 illustrates the following:

- The end station contains two independent physical NICs. Each NIC supports
  - One or more hardware-embedded VEPA. While this figure illustrates only one VEPA (blue or green) per NIC, an implementation may support multiple VEPA.
    - In this example, the blue and the green VEPA are completely separate, independent entities, i.e. they do not share any resources and cannot directly communicate with one another.
  - One or more physical ports attached to an adjacent bridge. A VEPA has only one logical uplink.

- In this example, each NIC supports direct I/O via PCI SR-IOV technology.
  - Direct I/O allows a VM to bypass the hypervisor and directly access the NIC to send / receive packets. Direct I/O is achieved by using a light-weight PCI Function called a Virtual Function (VF) to act as a conduit between the VM and the NIC hardware. This is analogous to a NIC supporting multiple traditional PCI Functions but is less hardware-intensive. Each VF is associated with a Physical Function (PF) which can be used by the hypervisor as a management conduit to provide overall control of the device or the port depending upon the implementation.

- Each VM may support one or more Virtual NICs.
  - In this example, each VM supports two vNIC – one per physical NIC.
  - Each vNIC contains a VSI which is associated with a SR-IOV VF to provide the direct I/O conduit.

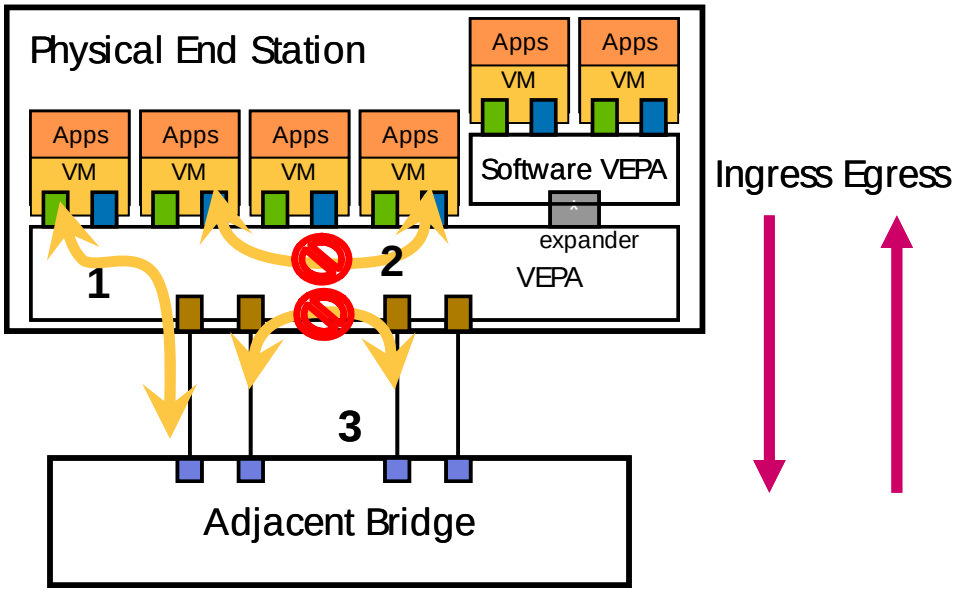


Figure 6. VEPA Frame Relay Support

As illustrated in Figure 6, a VEPA forwards packets as follows:

- VEPA forwards packets based on the MAC address and optionally via a port group or VLAN identifier.
- All VEPA traffic must be forwarded from the VSI to the uplink of an adjacent bridge (path 1).
  - VSI-to-VSI packet forwarding is not allowed (path 2).
- A VEPA supports only a single active logical uplink
  - Uplink-to-uplink packet forwarding is not allowed (path 3)
  - A VEPA may be partitioned into multiple logical VEPA each associated with its own independent uplink.
- A VEPA does not participate in or affect spanning tree operation, i.e. VEPA internal topology is not visible to the adjacent bridge, except for management associated TLVs (e.g. EVB TLV in this document).

Based on the prior materials, the reader should note the significant overlap in functionality and potential implementation between a VEB and a VEPA with the primary difference occurring in frame relay support. Further, this difference determines where and how network features are surfaced and their associated impact on system functional robustness and performance. This difference allows VEPA solutions to provide the following benefits:

1. Reduces complexity and potentially enables higher performance by off-loading advanced network functions from the VM or hypervisor to the adjacent bridge.
2. Allows NICs to maintain low cost circuitry by leveraging advanced functions on the adjacent bridge.
3. Enables a consistent level of network policy enforcement by routing all network traffic through the adjacent bridge with its more complete policy-enforcement capabilities.
4. Provides visibility of inter-VM traffic to network management tools designed for adjacent bridge.
5. Reduces the amount of network configuration required by server administrators, and as a consequence, reduces the complexity for the network administrator.
6. Can increase solution performance by off-loading advanced network functionality that may be computationally intensive to implement within a hypervisor or VM to the adjacent bridge.

As it can be seen, a VEPA provides a number of benefits but it too has limitations:

- Promiscuous support – To support a promiscuous VSI, a VEPA address table must be configured with all VM source MAC addresses. This requires either adding MAC address learning support or provisioning large address tables. Either option adds implementation cost and complexity.
- Support for simultaneous VEB, VEPA, and directly accessible ports on the same physical link – The adjacent bridge can only process a frame based on its contents and therefore lacks sufficient information to delineate between these three operating modes.
- Hierarchy of unrestricted physical ports – Normal bridge learning and flooding is not possible due to the lack of information within a frame.

To address these limitations, IEEE Std. 802.1ad-2005 is applied. This standard enables multiple virtual channels to be multiplexed on a single physical link – referred to as multi-channel functionality. Individual channels are delineated by a tag which is added to the frame and processed by S-VLAN Components (a bridge component) which are logically inserted into the adjacent bridge and the physical end station below the virtual bridge layer as illustrated in the following figure.

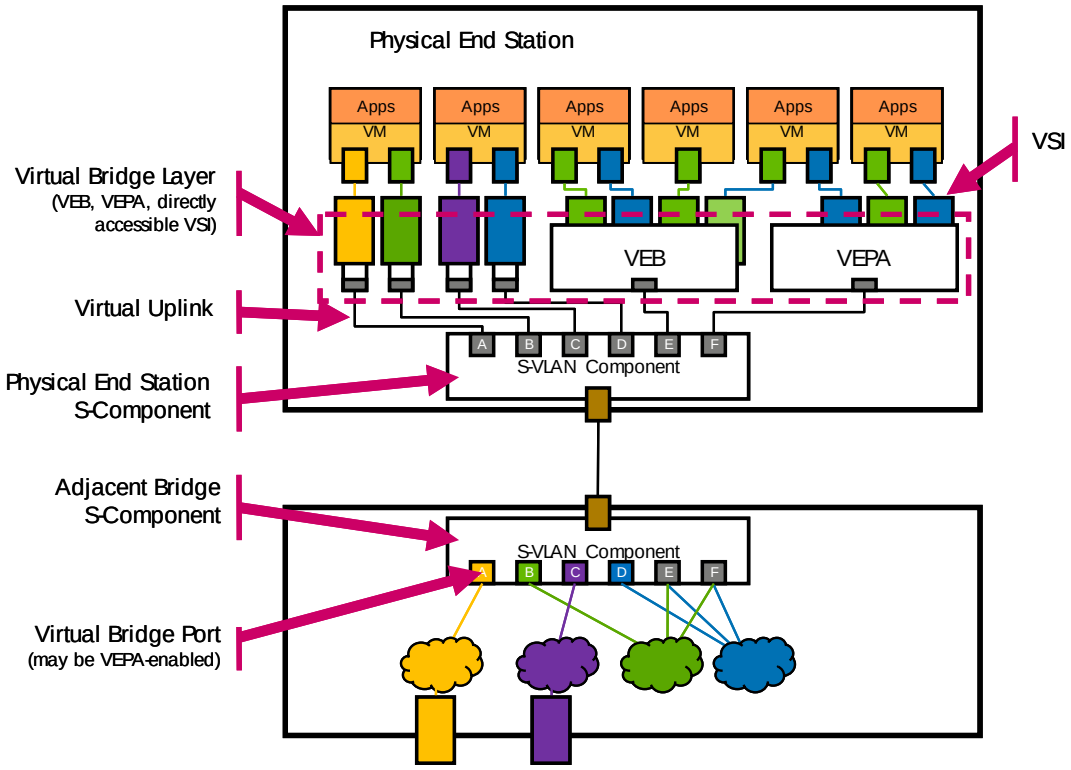


Figure 7. Mutichannel Ethernet Components

The S-VLAN component recognizes, inserts and removes service VLAN tags (S-Tag) to enable multiple channels in the bridged network. Adding an S-VLAN component to an end-station allows VEPA, VEB, and individual VSI to operate independently and simultaneously. Each VEPA, VEB, or individual VSI operates over its own virtual uplink instantiated by a pair of S-VLAN components - one in the adjacent bridge and one on the end-station.

The virtual uplinks created by the end-station's S-VLAN component are effectively connected over a multi-channel uplink to virtual ports (vPort) created by the S-VLAN component on the adjacent bridge as illustrated in the following figure.

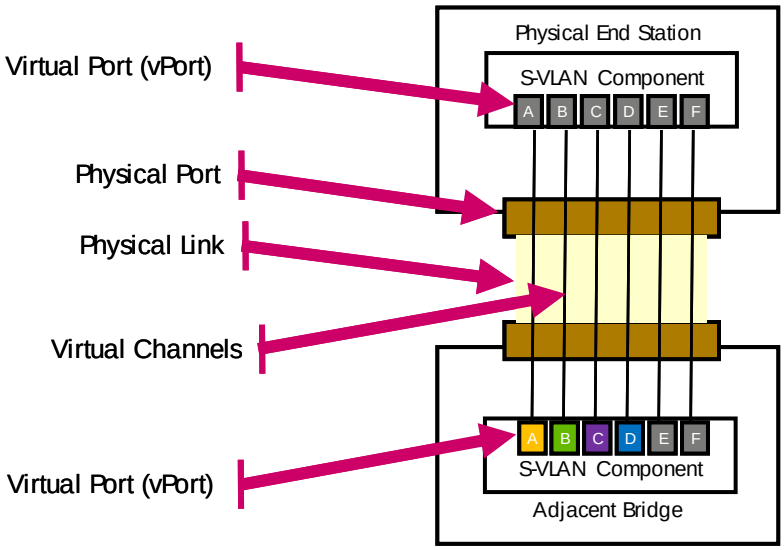


Figure 8. Multiple channels with on vPort at each end

Each frame traversing the physical multi-channel uplink will all have an S-Tag inserted by the first S-VLAN component it encounters and removed by the second S-VLAN component as it reaches the far side of the multi-channel link. The S-Tag inserted by the end-station identifies the particular source virtual uplink and the S-Tag inserted by the adjacent bridge identifies the destination virtual uplink. Any frames that must be broadcast or flooded to more than one VSI are replicated by the adjacent bridge and delivered across the multi-channel uplink as many times as needed, each with the proper S-Tag inserted.

Adding the multi-channel capability to the end-station solves the problem of supporting virtual machines needing promiscuous ports by isolating such VSI in a separate channel. By doing so, normal learning and forwarding behavior is pushed to the adjacent bridge, isolating it from the simple forwarding of the VEPA. It also allows the end station administrator to choose how virtual VM are connected to the network. A group of VM that require direct connectivity between each other for high performance and low latency can be attached to a VEB. Another group that requires traffic visibility, firewall inspection or other services on the adjacent bridge can be attached to a VEPA. Finally any individual VM that requires an isolated promiscuous VSI can be attached directly to a virtual uplink.

The subsequent chapters within this proposal provide additional details on VEPA and multi-channel operation, discovery, and configuration.

3. Architecture and Operational Overview

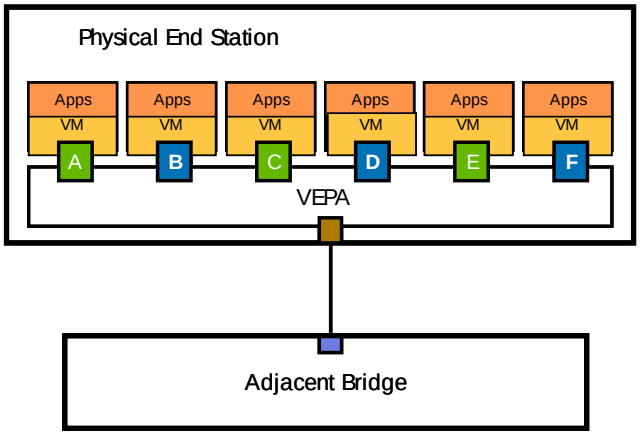
This chapter will describe VEPA and Multi-Channel architectural components and illustrate how these components are used via example operations.

3.1 VEPA Address Table Management

As a network edge end station, a VEPA is not required to support address learning. Instead, the VEPA address table is populated through a registration process. As an address, filter, or VLAN identifier is registered, the server virtualization infrastructure (e.g. the Hypervisor) updates the corresponding VEPA address table entry. This applies to:

- VSI default MAC address
- Locally Administered Address (LAA)
- Multicast addresses
- Promiscuous address mode support

The following figure illustrates an example physical end station, VEPA, and the associated VEPA address table.



Destination MAC	VLAN	Copy To (ABCDEF)
A	1	100000
B	2	010000
C	1	001000
D	2	000100
E	1	000010
F	2	000001
Broadcast	1	101010
Broadcast	2	010101
Multicast C	1	101010
Unknown Multicast	1	100010
Unknown Multicast	2	010101
Unknown Unicast	1	000000
Unknown Unicast	2	000000

Figure 9. Example VEPA with associated conceptual VEPA Address Table

In this example, the VEPA address table holds the following:

- A unicast MAC address (and VSI Instance Identifier) per VSI
- Per VLAN broadcast address – frames are forwarded to the VSI indicated by the corresponding bit mask
- Specified multicast addresses – VSI A, C, and E are the only participants in the specified multicast group.
- Per VLAN unknown unicast address – unknown frames are discarded on ingress, they are sent to the adjacent bridge on egress (i.e. from VSI Instance to bridge).
- Per VLAN unknown multicast address – unknown frames are forwarded to the VSI indicated by the corresponding bit mask

The address table is configured by the server virtualization infrastructure (e.g. the Hypervisor) simplifying the VEPA implementation by eliminating the need to support dynamic address learning. Further, the hypervisor can configure additional address table fields (not shown in the figure examples) such as QoS settings, VLAN configuration, promiscuous listening support, and so forth to provide additional functional capabilities.

3.2 Egress Processing

VEPA egress is defined as the set of operations required to transfer a frame from a VSI to the VEPA uplink. Since all frames are required to be forwarded to the uplink, the frame is moved from the VSI to the physical uplink for transmission and the S-VLAN-Tag is added.

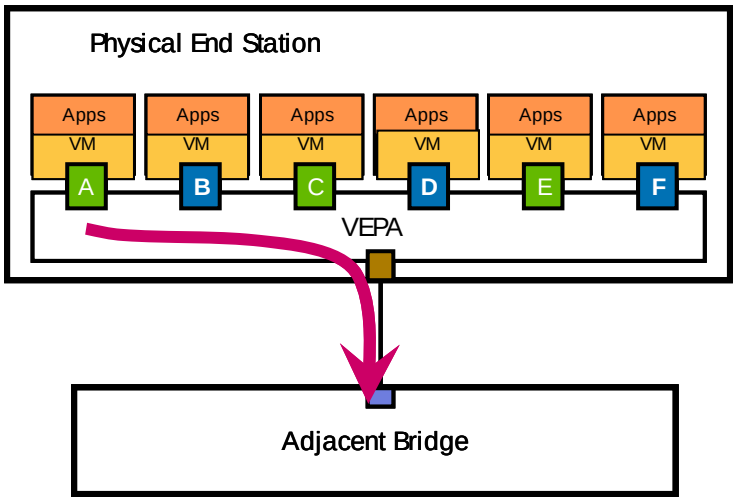
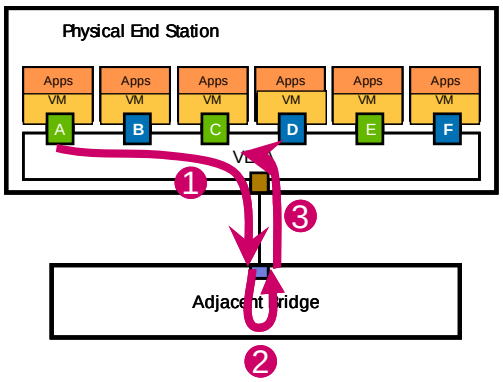


Figure 10. VEPA Egress Processing

While the main objective of VEPA is to forward frames to the adjacent bridge for advanced processing, VEPA implementations may provide additional processing on the frame since the VEPA comprehends the source VSI Instance. For example, source VSI MAC address validation to prevent spoofing, application of QoS and bandwidth management policies, VLAN formatting validation (tagged or untagged), and so forth. The adjacent bridge can only operate on the frame's contents (MAC address, VLAN ID, etc.) and is unaware of the source VSI Instance therefore eliminating such functional possibilities.

3.3 Ingress Processing

VEPA ingress is defined as the set of operations required to steer and transfer a frame received on the uplink to the appropriate VSI or set of VSIs. The VEPA must make use of the address table to perform this operation correctly. Address table access is illustrated in the following figure as VSI A transmits a unicast frame to VSI C.



VEPA Address Table

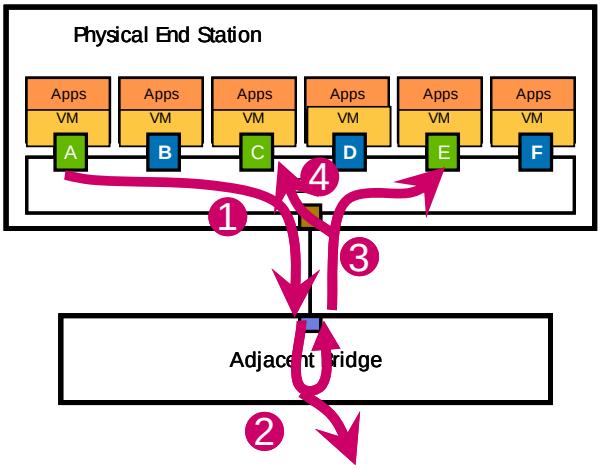
Destination MAC	VLAN	Copy To (ABCDEF)
A	1	100000
B	2	010000
C	1	001000
D	2	000100
E	1	000010
F	2	000001
Broadcast	1	101010
Broadcast	2	010101
Multicast C	1	101010
Unknown Multicast	1	100010
Unknown Multicast	2	010101
Unknown Unicast	1	000000
Unknown Unicast	2	000000

Figure 11. VEPA Unicast Ingress Processing from Source A to Destination C

In this example, the transmission and reception processing is:

1. VSI A performs egress processing and performs any additional functionality prior to the frame being transmitted out the egress port (step 1)
2. The adjacent bridge has enabled VEPA communication. The bridge applies the appropriate network processing to the frame and reflects the frame to the VEPA uplink (step 2).
3. Upon frame receipt and validation, the VEPA searches the address table to locate the destination VSI Instance based on the contents of the unicast frame (minimally the Destination MAC address and the VLAN Identifier). In this example, the "Copy to" mask indicates VSI C is the destination and the VEPA delivers the frame (step 3).
    a. If the unicast address is unknown, then the "Unknown Unicast" for the associated VLAN identifier would determine the appropriate "Copy to" mask, which is x000000 and the frame is discarded.

The address table access and associated processing is similar for multicast and broadcast with one exception. The originator of a multicast or broadcast frame may have been one of the VSI before the adjacent bridge reflected the frame back. In this case, the VEPA must perform additional filtering to avoid delivering the frame to its originator. This is illustrated in the following figure.



VEPA Address Table

Destination MAC	VLAN	Copy To (ABCDEF)
A	1	100000
B	2	010000
C	1	001000
D	2	000100
E	1	000010
F	2	000001
Broadcast	1	101010
Broadcast	2	010101
Multicast C	1	101010
Unknown Multicast	1	100010
Unknown Multicast	2	010101
Unknown Unicast	1	000000
Unknown Unicast	2	000000

Figure 12. VEPA Multicast Ingress Processing from Source A to Mulicate Group C

In this example, the transmission and receive processing is:

1. VSI A performs egress processing and performs any additional functionality prior to the frame being transmitted out the egress port (step 1)
2. The adjacent bridge has enabled VEPA communication. The bridge applies the appropriate network processing to the frame and reflects the frame to the VEPA uplink (step 2).
3. Upon frame receipt and validation, the VEPA searches the address table to locate the destination VSI based on the contents of the multicast frame (step 3). To prevent the frame from being delivered to its originator, the VEPA performs a source address lookup and filters out the VSI associated with the source address from the original "Copy To" mask associated with the destination address (step 3). The delivery mask is constructed via (Copy To = (Destination Copy To) AND !(Source Copy To). In this example,
   Destination Copy To = 101010
   Source Copy To = 100000
   Delivery Mask = 001010
4. The frame is replicated using the delivery mask (step 4).
   a. If the unicast address is unknown, then the "Unknown Unicast" for the associated VLAN identifier would determine the appropriate "Copy to" mask, which is x000000 and the frame is discarded.

3.4 Multi-Channel Operation

This section illustrates multi-channel operation through several example configurations. In these examples, an S-VLAN component is logically inserted into the adjacent bridge and the physical end station. Further, between these S-VLAN Components, six channels (A-F) have been established and associated with a directly accessible VSI, a VEB, or a VEPA.

The first example illustrates how a directly accessible VSI operates over a multi-channel configuration when communicating to a VSI accessible through the adjacent bridge.

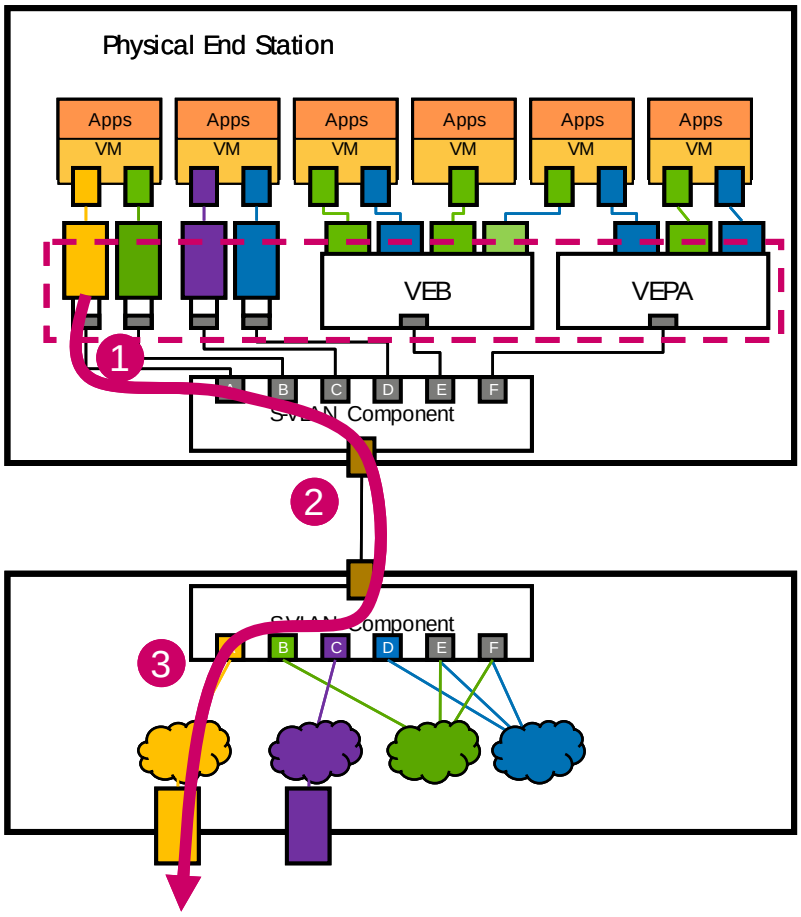


Figure 13. Frame fowarding from a directly accessible VSI over a multi-channel link

1. VSI A performs egress processing and performs any additional functionality prior to the frame being forwarded to the S-VLAN Component within the physical end station. (step 1)
2. The S-VLAN Component inserts an S-Tag associated with channel A into the frame and forwards the frame to the adjacent bridge. (step 2)
3. Within the adjacent bridge, the S-VLAN Component removes the S-Tag and forwards the frame. (step 3)

This example illustrates how a VEB operates.

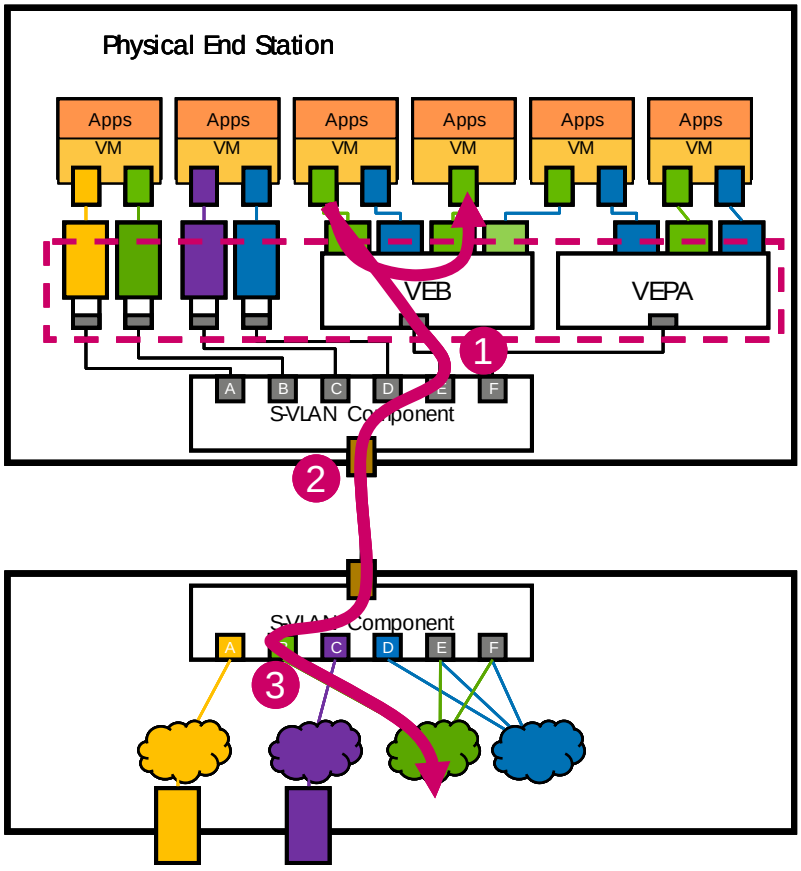


Figure 14. Frame fowarding when multi-channel is configured underneath a VEB

1. VM-to-VM communication across a shared VEB does not involve the multi-channel link.
2. The frame forwarding steps to communicate to a VSI not attached to the VEB are identical to the communication used for a directly accessible VSI running over a channel.
    a. VSI A performs egress processing and performs any additional functionality prior to the frame being forwarded to the S-VLAN Component within the physical end station. (step 1)
    b. The S-VLAN Component inserts an S-Tag associated with channel A into the frame and forwards the frame to the adjacent bridge. (step 2)
    c. Within the adjacent bridge, the S-VLAN Component removes the S-Tag and forwards the frame. (step 3)

This example illustrates VM-to-VM communication through a VEPA when multi-channel is configured.

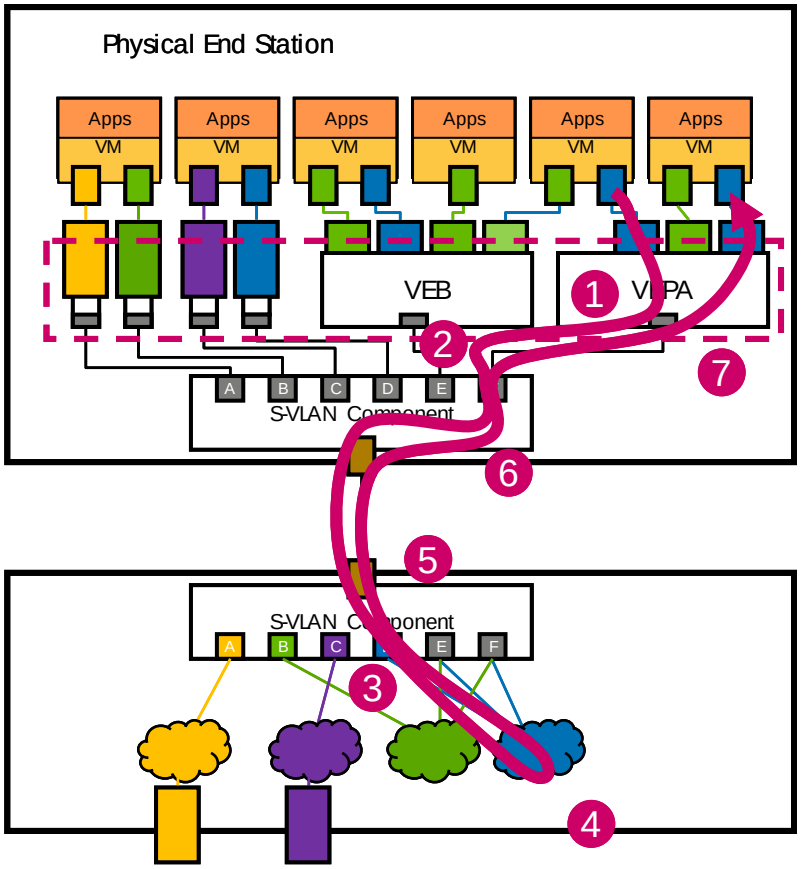


Figure 15. Frame fowarding when multi-channel is configured underneath a VEPA

1. VSI performs egress processing and performs any additional functionality prior to the frame being forwarded to the S-VLAN Component within the physical end station. (step 1)
2. The S-VLAN Component inserts an S-Tag associated with channel F into the frame and forwards the frame to the adjacent bridge. (step 2)
3. Within the adjacent bridge, the S-VLAN Component removes the S-Tag and forwards the frame (step 3).
4. The adjacent bridge determines that the vPort is configured for VEPA mode so it forwards the frame based on the bridge forwarding table (step 4).
5. Within the adjacent bridge, the S-VLAN Component adds the S-Tag associated with channel F and forwards the frame to the S-VLAN Component within the physical end station (step 5).
6. The S-VLAN Component within the physical end station removes the S-Tag and forwards the frame to the associated VEPA (step 6).
7. The VEPA forwards the frame based on its VEPA address table to the associated VSI (step 7).

The following example illustrates how a VEPA-attached VM communicates to a directly attached VSI through a common physical end station.

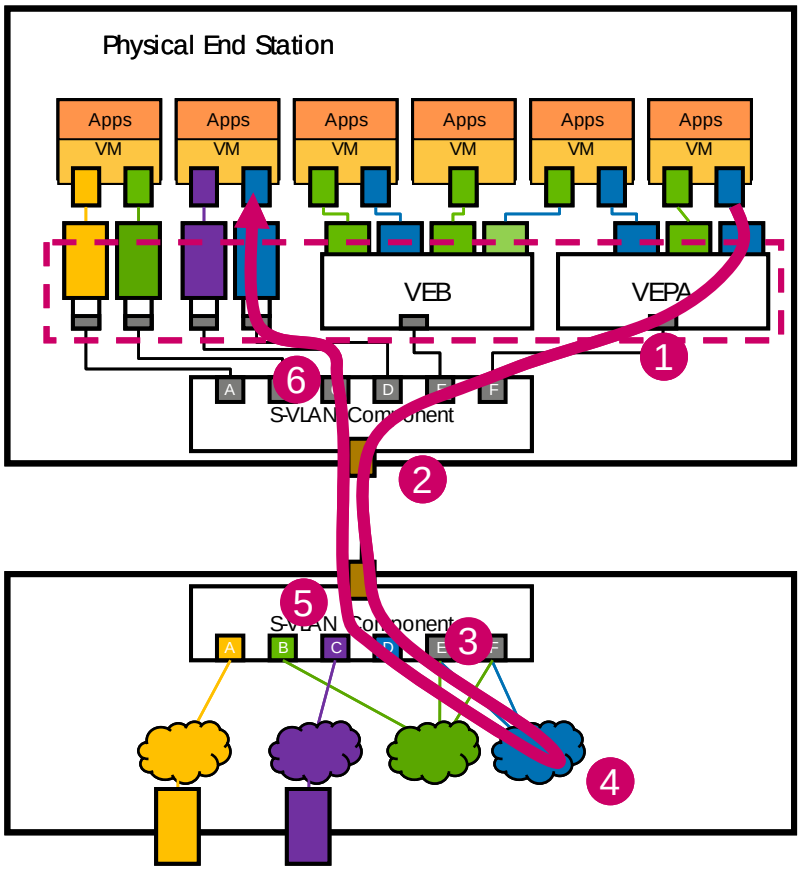


Figure 16. Frame fowarding over multi-channel between a VEPA and adirectly attached VSI

1. VSI performs egress processing and performs any additional functionality prior to the frame being forwarded to the S-VLAN Component within the physical end station. (step 1)
2. The S-VLAN Component inserts an S-Tag associated with channel F into the frame and forwards the frame to the adjacent bridge. (step 2)
3. Within the adjacent bridge, the S-VLAN Component removes the S-Tag and forwards the frame (step 3).
4. The adjacent bridge determines that frame's next hop is associated with channel D and forwards the frame to the S-VLAN component.
5. Within the adjacent bridge, the S-VLAN Component adds the S-Tag associated with channel D and forwards the frame to the S-VLAN Component within the physical end station (step 5).
6. The S-VLAN Component within the physical end station removes the S-Tag and forwards the frame to the directly attached VSI.

3.5 Edge TLV Transport Operation

Today, IEEE control plane discovery operations are performed over unacknowledged protocols, such as LLDP and DCBX. The Edge TLV Transport (ETTP) provides

acknowledgements, which signal to the sender that the receiver is able to receive an additional ETTP Data Unit. ETTP enables the sender to transmit discovery operations more frequently than would be the case with timer based approaches. The intent is to have the server's virtualization infrastructure (e.g. Hypervisor) implement ETTP, versus having the NIC implement ETTP.

The following diagram depicts, at a high level, ETTP semantics. In step 1, the ULP passes an outgoing ULP Data Unit to ETTP by invoking a transmit request procedure. In step 2, the ULP Data Unit, which for some ULPs (e.g. VSI) may contain a set of ULP TLVs, is transmitted and a ETTP low level Acknowledgement (L-ACK in the diagram) timer is set, but the frame is not yet deleted from the transmit buffer until a ETTP is received for that ETTPDU. In step 3, the arriving ETTP frame is received into a receive 'buffer', where it is held until it is removed by a T3 indication procedure that passes the ULP Data Unit to the associated upper level protocol. In step 4, when the receive buffer is emptied, a low-level acknowledge (L-ACK) is sent to the sender. In step 5, if the L-ACK is received before the L-ACK timer expires, then the transmit buffer is cleared and ETTP can process another ULP PDU through the ETTP procedure. However, if the L-ACK timer expires before the L-ACK is received, then the frame in the transmit buffer is resent (some preset number of times).

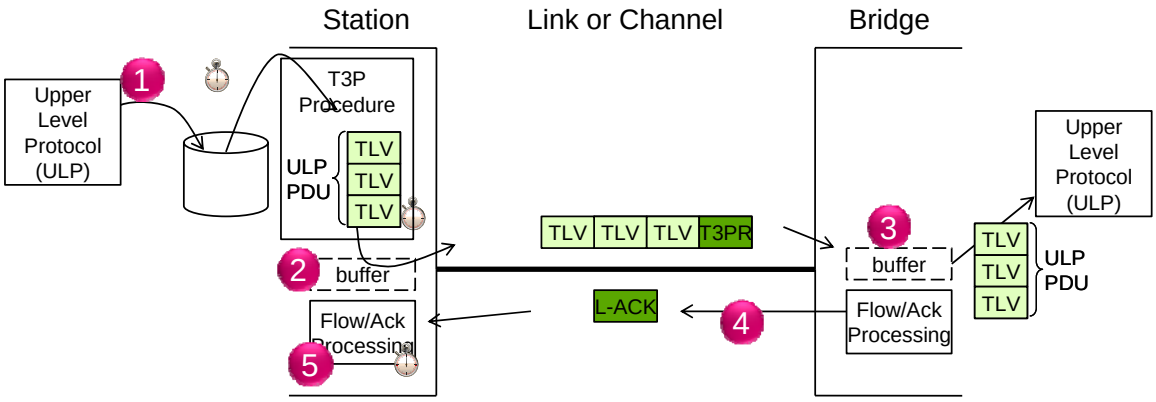


Figure 17. Example ETTP Exchange

3.6 VSI Discovery and Configuration Protocol (VDP) Operation

Enterprise and cloud-based networking deployments have been rapidly growing in size leading to a significant increase in the complexity of Ethernet networking in data centers. The advent of virtualization technology brings unprecedented network configuration complexity due to the significant increase in the number of Ethernet switches and very large number of Virtual Station Interfaces (VSIs). Problem is made more complex by advent of Virtual Machines (VM) mobility and solutions requiring external network state to move with the VM, when the VM moves.

Virtual Ethernet Bridges (VEBs) embedded in Virtualized Systems have been around for decades. VEBs provide efficient VM to VM communications. However, today's virtual

switch management is too manual and x86 scale-out server sprawl and virtualization magnifies this complexity. Two of the major challenges associated with today's virtualization approaches is the ability to automate the association of a VSI Instance with it's network state and automate VM migration, including all the network state associated with the VM.

Today, when a VM moves from one server to another, VEBs embedded in Virtualized Systems migrate the internal VEB's VSI Type that is associated with the VM. The VSI Type consists of the network state associated with the VM and may include Access and QoS Controls. In today's implementations, the external switch's port profiles do not move with the VM. Client have three options for dealing with this issue. Option 1 is to use the same VSI Type for all VMs, the problem with this approach is that it limits virtualization's value, because all VMs in the network must be doing the same type of work (e.g. all must be e-mail VMs). As a result, if a group of servers doing the same type of work gets over utilized, the VMs from those servers cannot be moved to a group of servers doing another type of work (e.g. file/print).

Option 2 is to move the VSI Type after the VM moves. This can be done by having the external switch look up the VSI Type when the VM starts sending messages on the new server. For example, when the VM starts sending messages, the external switch uses the VM's MAC Address to look-up the port profile. This approach suffers from two problems: The external switch cannot tell if the MAC Address used by the VM is a migrated MAC address (i.e. from a migrated VM) or a re-incarnated MAC address (i.e. from a new VM that is using a previously destroyed VM's MAC address). The second problem is that there is a VSI Type exposure window between the VM's first message and the time it takes the external switch to obtain the VSI Type from the switch's fabric manager.

Option 3 is to simply configure the link between the server and the edge switch as a trunk port. The issue with this approach is all physical servers must be in the same security domain, which has the similar VM movement limitations as option 1. For example, a physical server cannot be managed by tenant A in the same fabric as a physical server that is managed by tenant B.

The VDP Protocol specified in this document enables the association of a VSI Type with a VSI instance (e.g. a VM virtual port) and the de-association of a VSI Type with a VSI instance (e.g. a VM virtual port). VDP simplifies and automate Virtual Server (VS) network configuration by enabling the movement of the VSI Type when the VSI Instance moves.

3.6.1 VDP Type Configuration and Automation

A virtualized server hosts a set of VMs. Each VM may support one or more Virtual Station Interface (VSI) Instances. Typically, a VM will support a virtual NIC (vNIC) that emulates a physical NIC. Each vNIC will contain a VSI which is connected to a VEB or VEPA. The server's virtualization infrastructure (e.g. a Hypervisor) assigns one or more VSIs to a VM to access the network. The VM is able to communicate with other VMs on the same physical server through the VSI Instance. Similarly, the VM is able to communicate to external stations through the VSI Instance.

Each VSI Instance is assigned VSI Type ID (VTID). VSI Type definition is outside the scope of this proposal. For information context purposes only, a VSI Type definition may include port access or rate limiting controls. Prior to the activation of a VM, VDP exchanges are used to associate a VSI Instance with a VLAN Identifier, a MAC Address and a VTID in the adjacent bridge and, if VEB is used, VEB. Similarly, a VDP exchange is used to de-associate a VSI Instance with a VLAN Identifier, a MAC Address and a VTID in the adjacent bridge and, if VEB is used, VEB, when a VM is either destroyed or moved.

The following sections provide an operational overview of how VDP can be used to automate the configuration of network state (e.g. VSI Type) and the association of network state to a VSI Instance. It will then describe the management elements required to support such an example.

3.6.1.1 VDP – Operational Example

An example of the steps associated with VDP is depicted in the following figure.

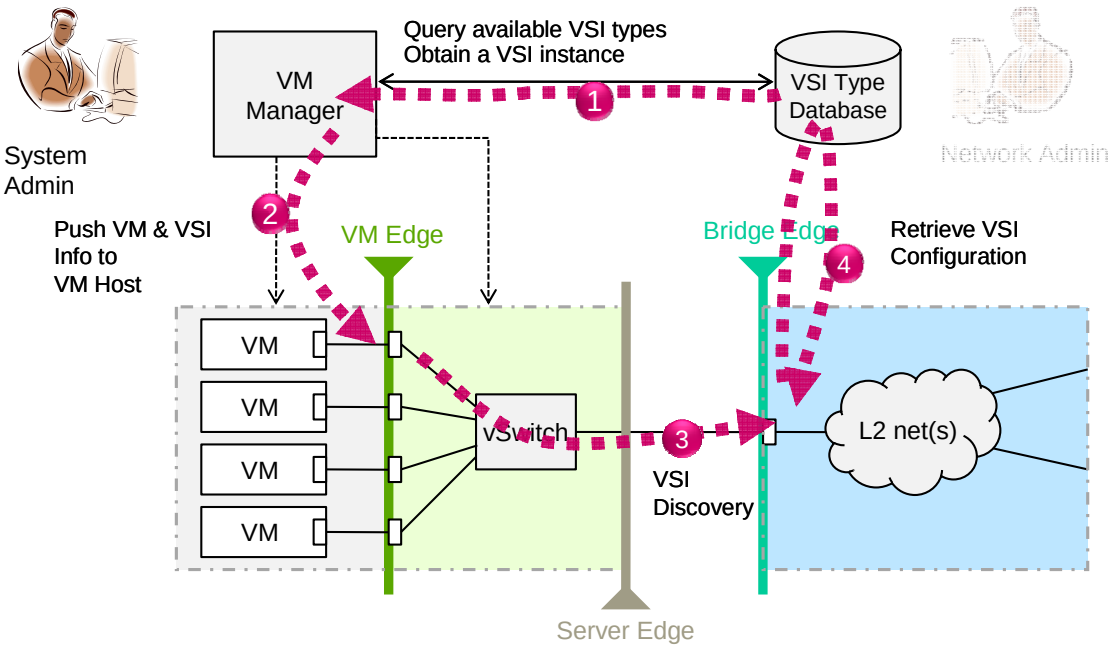


Figure 18. VSI Type Architectural and Operational Overview

Following are the steps depicted in the figure above:

Step 1: VM Manager queries available VTIDs and creates a VSI Instance consisting of VSI Instance ID and the chosen VTID. The VTDB server may create and track VSI Instance.

Step 2: VM Manager configures VSI with VTID and VSI Instance ID obtained from VTDB.

Step 3: Before VSI Instance (VM) activation, the VDP Module performs VSI Discovery and Configuration protocol exchanges to associate the VSI instance with a VTID, MAC Address and VLAN Identifier. In this example approach, the VDP Module is implemented as part of the server's virtualization infrastructure (e.g. in the Hypervisor or a service VM guest running on top of the Hypervisor). The VDP Module is also implemented in the adjacent bridge.

Step 4: As part of the VDP exchange the adjacent Bridge retrieves the VSI Type from the VTDB by using the VTID and possibly the VSI Type Version and VSI Instance. The adjacent Bridge stores the association of VLAN ID, VSI Type, VSI Type Version and MAC Address in its local memory. This association is then applied to the traffic flow from/to the VSI Instance. Note the VTDB access protocol is not part of this document.

3.6.1.2 VSI Type Database (VTDB)

The VSI Type Database described above is used to store detailed definition of VSI types. Again these definitions are outside the scope of this document. For information purposes only, a VSI Type may contain access and traffic controls. Also for information purposes only, a VSI Type Database is expected to be part of the database used by the edge switch's Network Change and Configuration Manager.

VSI Type Definitions within a VTDB are identified by VSI Type ID (VTID) and VTID version. Optionally, VSI instance specific definitions are possible.

The mechanisms used to create VSI Types in a VTDB are outside the scope of this document. For information purposes only, each VSI Type may refer to different use models, such as a server type, where each server type (e.g. web, file/print, e-mail) has a unique VSI Type. Many other use models are possible.

3.6.2 VSI Type Definition and Management

VSI Type Definition and Management is outside the scope of this document. In other words, the content of a VSI Type entry in the VTDB and how that content is managed are outside the scope of this document.

Similarly, VSI Type management and access protocols are outside the scope this proposal. This is not a hindrance to deployment of VDP because current Data Center Network (DCN) infrastructure includes mature tools for management and configuration and can be easily deployed to manage VSI Types. Further, VSI Type Management

approach proposed in this document matches well with currently deployed DCN management practices. It is achieved by aligning management and configuration responsibility with current organization structure e.g. VSI Types can be managed by Network Administrator and deployed on servers by server administrators.

3.6.3 VSI Manager ID

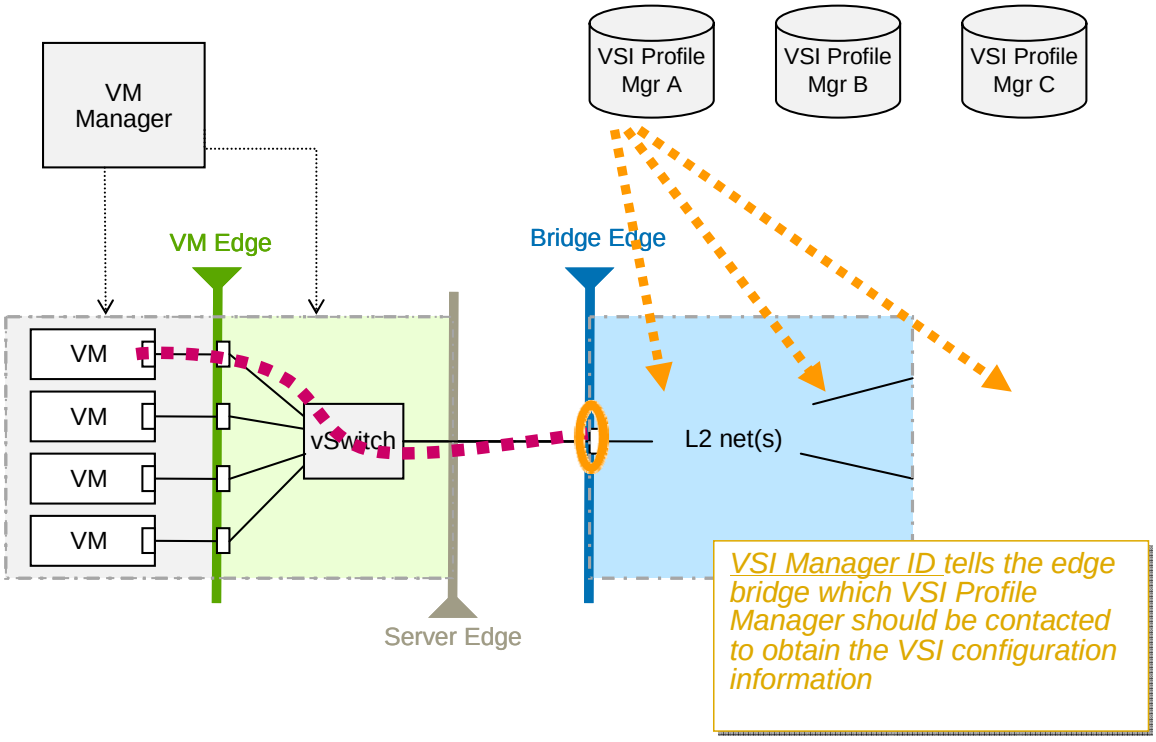


Figure 19. VSI Manager ID

VSI Manager ID tells the edge bridge which VSI Type Manager should be contacted to obtain the VSI configuration information. The VSI Manager ID is part of VDP exchange between Station and the Edge Bridge.

Note, the VSI Type ID or VSI Instance ID can also be used as index to look up VSI Type configuration in VSI Type Database, see the following figure:

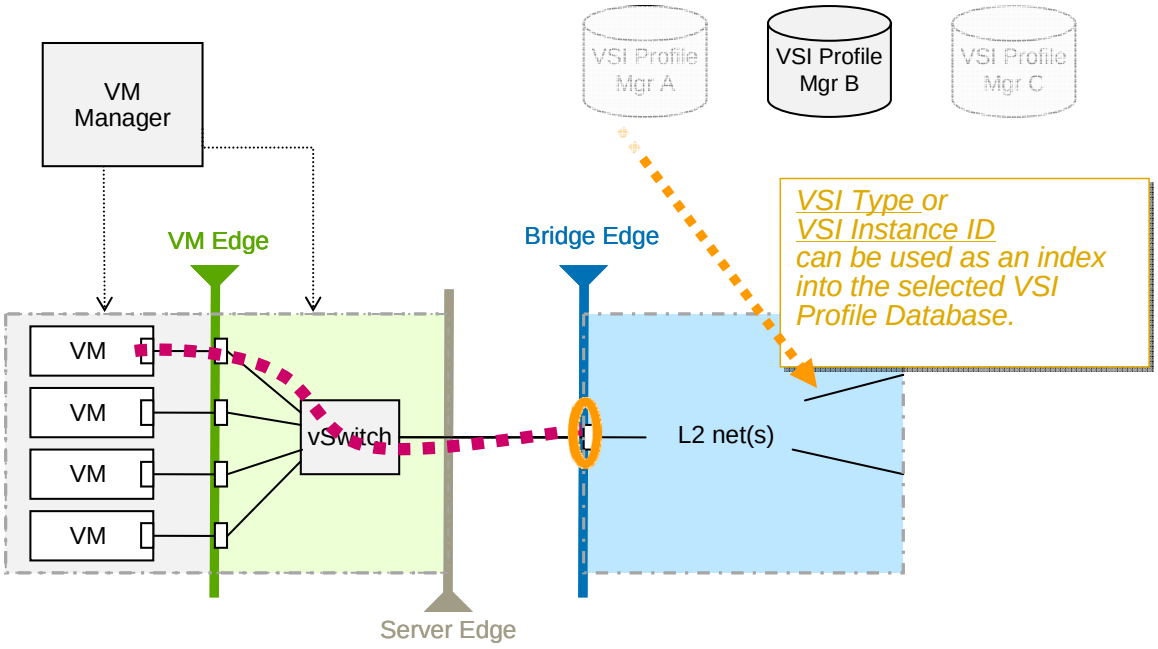


Figure 20. VSI Type or Instance ID

3.6.3.1 VSI Manager ID Usage Example

The VSI Manager ID Identifies the VSI Manager with the Database that holds the detailed VSI Type and/or VSI Instance Identifier definitions. The contents of the VSI Manager Database are outside the scope of this proposal. The VSI Manager Database may use a combination of the following fields to index into the VSI Manager Database:

- VSI Type Identifier
- VSI Type Version
- VSI Instance Identifier

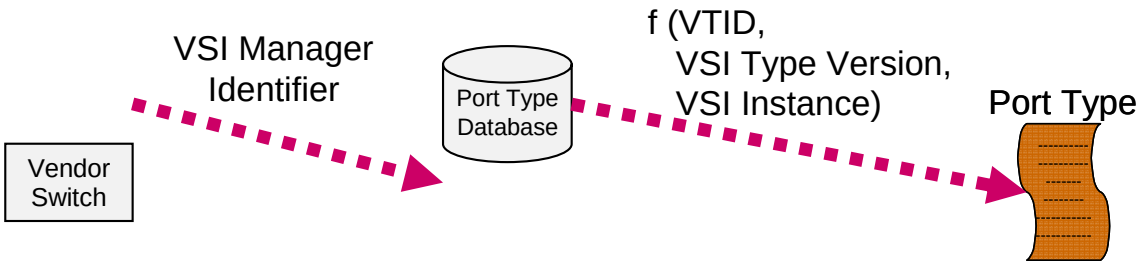


Figure 21. VSI Manager Database Lookup

4. Ethernet Virtual Bridging TLV Semantics

The EVB TLV is used to:

- Advertise a station or bridge's EVB functional and resource capabilities
- Activate common functional capabilities
- Reduce resource capabilities to a maximum common value

The EVB TLV is exchanged via LLDP and conforms to the LLDP TLV specification. The EVB TLV is illustrated in the following figure:

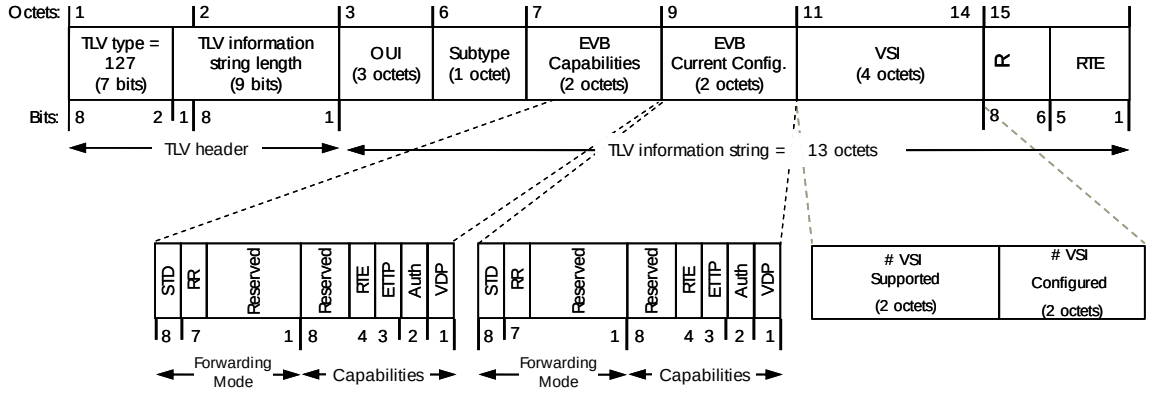


Figure 22. EVB TLV Format

The EVB TLV fields are:

EVB Capabilities - The TLV describes EVB capabilities that are supported by the sender. The capabilities are:

- Forwarding Mode:
  - Standard 802.1Q forwarding
  - Reflective Relay – enables frames to be reflected back through the ingress port. For example, in a VEPA solution, frames exchanged between co-located VM must flow through the adjacent bridge. Reflective relay allows these exchanges to flow through a common uplink between the station and the adjacent bridge.
    - From the station, RR = TRUE indicates the station requests reflective relay support.
    - From the adjacent bridge, RR = TRUE indicates the bridge supports reflective relay support.
    - If the station and the adjacent bridge set RR = TRUE, then reflective relay can be enabled. The EVB TLV Current Configuration RR bit is set to TRUE.
    - If either side does not set RR = TRUE, the reflective relay cannot be enabled. The EVB TLV Current Configuration RR bit is set to FALSE.

- Retransmission Timer Exponent (RTE) – Indicates the current RTE value is present

- Edge TLV Transport Protocol (ETTP) – Indicates the sender supports ETTP

    - From the station, ETTP = TRUE indicates the station supports VDP.
    - From the adjacent bridge, ETTP = TRUE indicates the bridge supports VDP.
    - If the station and the adjacent bridge set ETTP = TRUE, then ETTP can be enabled. The EVB TLV Current Configuration ETTP bit is set to TRUE.
- If either side does not set ETTP = TRUE, then ETTP cannot be enabled. The EVB TLV Current Configuration ETTP bit is set to FALSE.

- 802.1X Authentication Required – Indicates the sender's software requires 802.1X authentication before applications can be network enabled

- VSI Discovery Protocol (VDP) – Indicates the sender supports VDP. VDP is dependent upon ETTP being enabled.
    - From the station, VDP = TRUE indicates the station supports VDP.
    - From the adjacent bridge, VDP = TRUE indicates the bridge supports VDP.
    - If the station and the adjacent bridge set VDP = TRUE and ETTP == TRUE, then VDP can be enabled. The EVB TLV Current Configuration VDP bit is set to TRUE.
    - If either side sets VDP = FALSE or ETTP == FALSE, then VDP cannot be enabled. The EVB TLV Current Configuration VDP bit is set to FALSE.

EVB Current Configuration – The TLV describes the EVB capabilities that are currently configured at the sender. Current configuration represents the intersection of the capabilities and resources between the two senders on a link.

- Number of VSI Supported – The maximum number of VSI that can be supported by the sender.

- Number of VSI configured – The maximum number of VSI that has been configured by the sender.
    - From the station, it indicates the number of resources that should be reserved by the adjacent bridge.
    - From the adjacent bridge, it indicates the number of active VSI discovered and configured.

- Retransmission Exponent (RTE) – RTE is an EVB link or channel attribute used to calculate the minimum ULP PDU retransmission time. The ULP PDU retransmission time is calculated as follows:
    - The Retransmission Granularity (RTG) is set to 10 micro-seconds.
    - The Retransmission Multiplier (RTM) is set to 2^{RTE}
    - The sender's ULP transmission timer is set to RTM * RTG
    - Both sides agree to the largest common value

The following illustrates an example EVB TLV exchange between a station (e.g. a hypervisor) and the adjacent bridge. This exchange is accomplished using LLDP. In this example, both the station and the bridge support Reflective Relay, VDP, and a set of VSI resources.

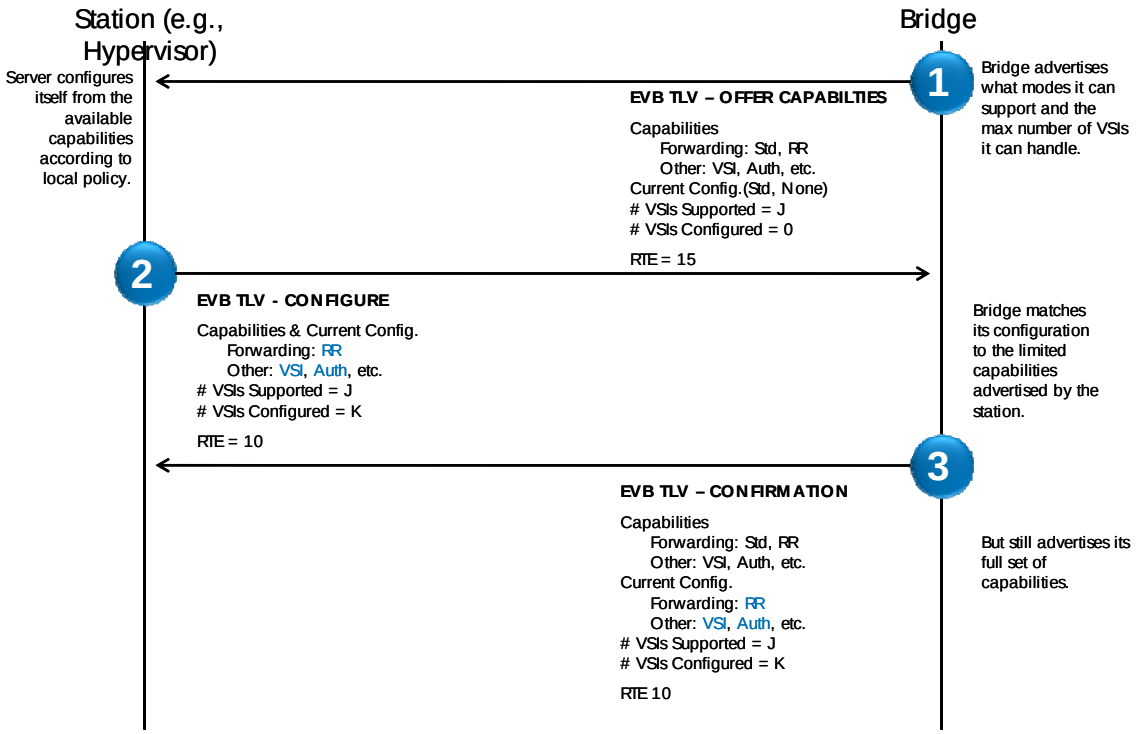


Figure 23. Example EVB TLV Exchange

5. Multi-Channel TLV Semantics and State Machine

This chapter provides an overview, detailed semantics, and state machines for the Mulit-Channel Discovery and Configuration Protocol (MDP).

5.1 MultiChannel Bridge Components and Operation

5.1.1 Introduction

The purpose of MDP is to configure S-VLANs (channels) used by a station to simplify the internal configuration and operation of Virtual Station Interfaces (VSIs), Virtual Ethernet Bridges(VEBs) and Virtual Ethernet Port Aggregators (VEPAs). S-VLANs are implemented in stations and bridges using a specialized S-VLAN aware bridge component. This component conforms to the Port-mapping S-VLAN Component specified in 802.1Qbc.

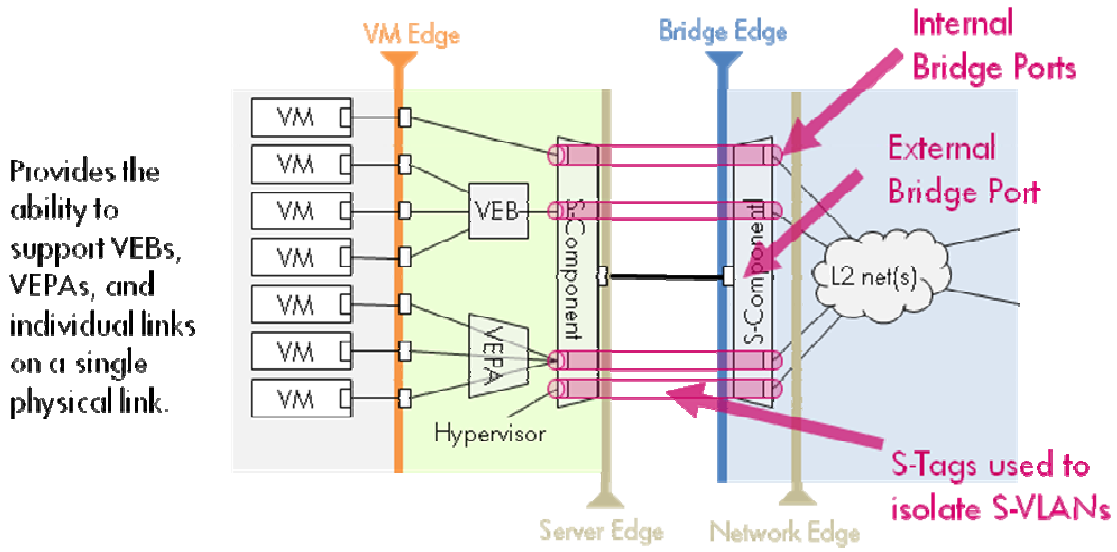


Figure 24. Example Multi-channel Block Diagram

The C-VLANs carried and the reflective relay operative mode associated with each S-VLAN is determined by the configuration of the Bridge. The configuration of the Bridge is determined by its capabilities and by requests made MDP described here and using the EVB TLV described in clause 4. The station's and Bridge's configurations are exchanged using LLDP TLVs. The configuration of S-VLANs is determined by an exchange of an LLDP TLV at the LAN level of operation. One LLDP databases exists for each LAN connecting between the station and Bridge. The reflective relay operation is then determined by a separate LLDP TLV exchange which occurs on top of the S-VLAN (see EVB TLV).

5.1.2 S-Component

The figure below is a "baggy pants" Bridge relay architecture model for the station and Bridge. The S-Component in this relay conforms to the Port-mapping S-VLAN component specified in 802.1Qbc. The S-Component is used to create S-VLANs (channels). The C-Component of the Bridge is a standard Bridge C-Component relay with the exception of additions for the reflective relay feature and support for EVB and VSI discovery, configuration and control.

Not all the represented components need to be present in an implementation. If MDP is present then the E-Components depicted will be present and are the operative parts for forming E-VLANs. If no MDP is present then the E-Components may be present, however disabled or may not be present at all. It is also possible that one or both of the E-Components will be absent. If no VEB or VEPA is present (no V-Comp) then the E-Component will couple directly to the end station LLC at the far left of the figure. It is possible to have the E-Component without at V-Component or a V-Component without an E-Component.

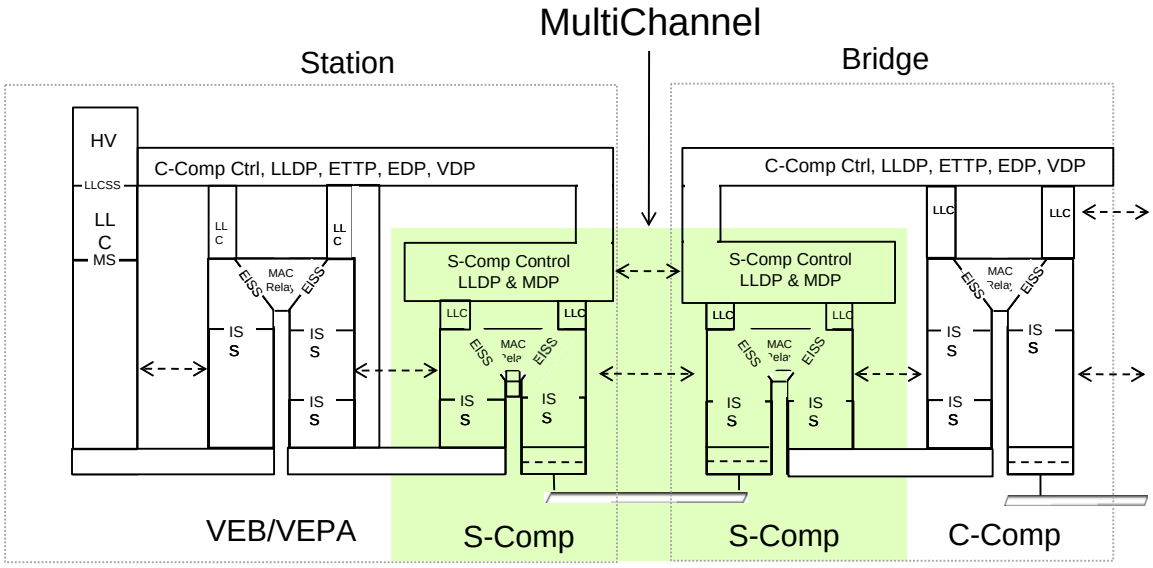


Figure 25. Station and Bridge V-Component and E-Component Block Diagram

MultiChannel is implemented using the S-Components, which form S-VLAN, along with configuration of the Bridge side C-Component. The C-VLANs carried by each S-VLAN are determined by configuration of the C-Component within the Bridge. Each S-VLAN is connected from a single internal S-Comp Bridge Port on the station to a single internal S-Comp Bridge Port facing an internal LAN within the Bridge. The internal LANs within the Bridge each span between one S-Comp internal Bridge Port and one C-Comp internal Bridge port.

The C-VLAN configuration and reflective relay configuration of the Bridge is determined by the configuration of the C-VLAN aware component of the Bridge.

5.2 MDP Discovery and Configuration

Multichannel is configured by the exchange of LLDP TLV at the LAN level. The exchange begins when the system is initialized. The configuration protocol begins with the station, which makes a request for channel resources from the Bridge. In response the Bridge provides the best matching set of S-VLANs it is capable of providing. It is possible the Bridge does not have all the resources requested in which case the Bridge response will provide a subset of the requested S-VLANs.

After initialization it is possible for the station to change it's multichannel configuration. The Bridge seeing a change in the stations request will alter it's configuration to match the needs of the station.

5.2.1 MDP TLV

The station and Bridge both use the same LLDP TLV to configure multichannel. This TLV is in LLDP OUI format (802.1AB sub-clause 8.6). The MultiChannel capabilities, requests and running configuration is encoded in the info field of this TLV as follows:

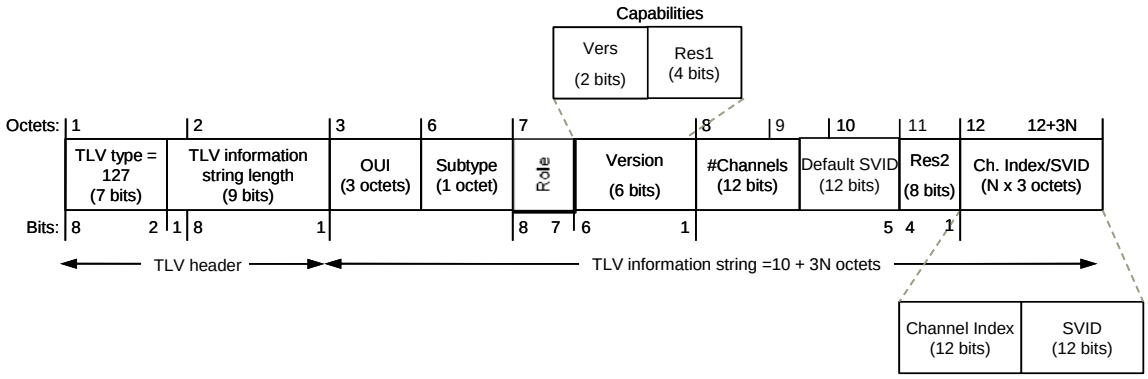


Figure 26. MDP TLV

- **Role** Bits (see note 3 regarding ties)
  - S(01b) – Indicates the sender assigns channels numbers and a default SVID for the default channel 1 and requests SVID assignments from the neighboring 'B'.
  - B(10b) – Indicates the sender accepts multichannel configuration requests from its neighboring 'S' and that the sender will do the best it can to fill the SVID assignment requests from the neighboring 'S'.

- **Version**– Describes multichannel capabilities that can be supported by the sender.
  - Vers: 10b identifies this version, 00b disables MCh
  - Res1: must be set to zero, ignored on receipt
- **# Channels** Supported – Identifies the number of SVID channels that are supportable by the sender.

- **Default SVID** – Reserved for future use.

- **Res2** – must be set to zero, don't run if non-zero

- **Ch#/SVID** Pairs
  - Channel # -- indicates the index number of the channel. The 'S' assigns channel numbers in the range 0-167. Zero is reserved. Channel number 1 is the default channel and is always the first channel in the list of pairs. The channel index should be between 1 and the maximum number of channels supported by the port which is indicated by the # channel field of the TLV.
  - SVID – The S-Tag VLAN ID assigned to the channel. The 'B' assigns SVIDs to channels in the range 1-0xffe. A 'S' uses the 0 SVID to request an SVID assignment from the 'B'.

Note1: A maximum of 167 channels can be supported. Other formats (assuming sequential SVIDs) could be defined to allow support for 2K+ channels.

Note2: This listing could be sparse (in order to indicate arrival and removal of channels). The channel going away is recognized by that channel index/SVID pair is removed.

Note3: If we have a tie, two 'B's or two 'S's MultiChannel the MDP protocol will not run. In the case of two Bridges, then one must take the S role while the other takes the B role.

Note4: The order of the list will determine the priority of SVID assignments. If the Bridge does not have resources for all channels it will assign the first channels in the list.

5.2.2 MDP Configuration Procedures

The MDP protocol used to discover and configure S-VLANs begins by announcing the presence of MDP along with the station and Bridge capabilities (1). After the initial announcement the Bridge will look for a request from the station (1). Once the Bridge sees a station request it will configure itself with and provide the best matching configuration to the station (2). The station seeing that the Bridge is now configured goes operative using the Bridge's configuration (3).

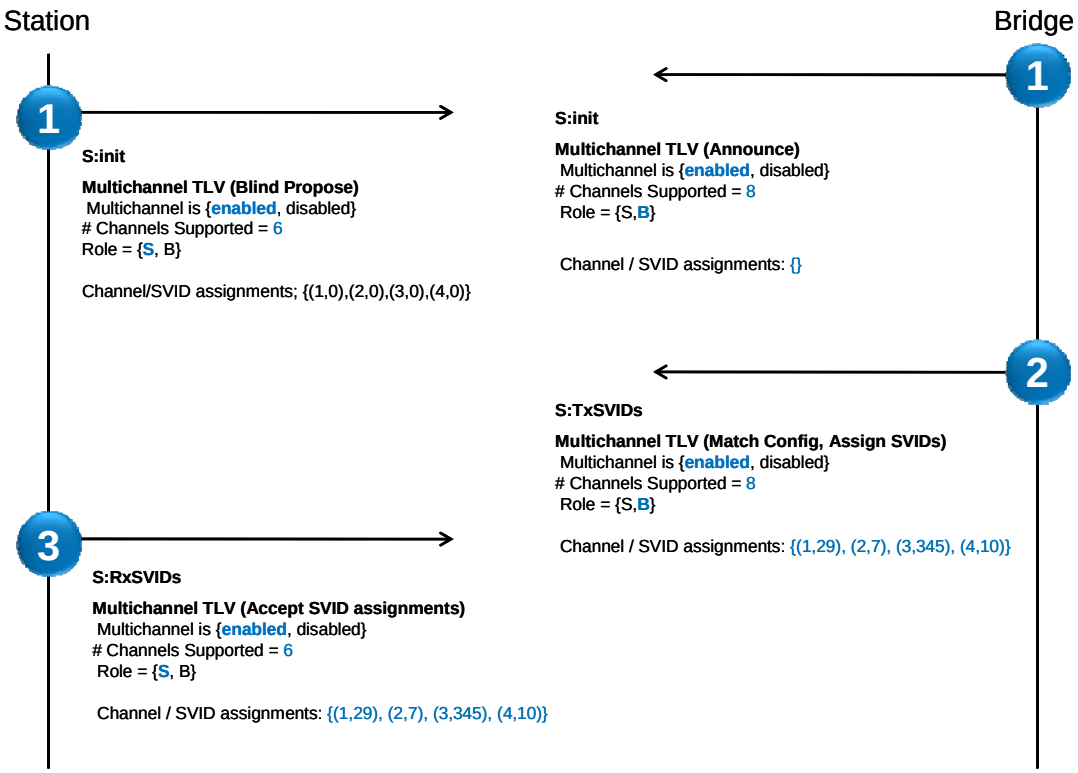


Figure 27. Example MDP TLV Exchange

5.2.3 MDP Configuration Variables

The following variables are used by the MDP Configuration state machine to perform multichannel configuration. The MDP requires each side of the configuration be assigned a role as a Bridge or a Station. This is done by setting the AdminRole variable. In most pieces of equipment the station or bridge role will not be settable, though the protocol allows for equipment which can take either role. For MDP to configure multichannel one side must take the station role and one side must take the Bridge role. If both sides of the LAN have equipment configured as stations or as bridges the protocol will not configure multichannel.

- AdminRole: Is the administratively configured value for the local port's role parameter. The value of AdminRole is not reflected in the MCh TLV. The AdminRole may take the value S or B. S indicates the sender is unwilling to accept multichannel configuration (mode, # channels supported, channel index) from its neighbor and that the sender is willing to accept SVID assignments from the neighbor. Stations usually take the S role. B indicates the sender is willing to accept multichannel configuration (mode, # channels supported, channel index) from its neighbor and that the sender is willing do the best it can to fill the SVID assignments from the neighbor. Bridges usually take the B role.
- OperRole: The current operational value of the Role parameter in the local port. This value is included as the Role parameter in the MCh TLV and may take values S or B as described for AdminRole.

- RemoteRole: Indicates the value in the remote MCh TLV role field. rwNull indicates either the TLV was not present in the last LLDP PDU or that no LLDP PDUs have been received. rwS and rwB indicate that the Role field was set in the MCh TLV received and that it had a value of S or B respectively as described for the AdminRole variable.
- mchState: The current running state of MultiChannel. The values for this variable are NOTRUNNING or RUNNING.

- AdminVersion: The administratively configured value for the MCh capabilities parameters. This value is included as the MCh Cap parameter in the MCh TLV. If the value is DISABLE = 000b it means MCh is disbled. If the value is VER0 = 100b it means this version.
- AdminChnCap: The administratively configured value for the Number of Channels supported parameter. This value is included as the # Channels supported parameter in the MCh TLV.
- AdminSVIDWants: The administratively configured value for (channel,SVID) pairs wanted by a S. Not used by a B. The value NONE means no channels are wanted. The channel numbers may be any valid number from 1-0xffe. A 0 channel number may be used to reserve space in a TLV. The SVID values are 0 indicating the S is requesting an SVID assignment from the 'B'. This value is used to form the (channel,SVID) pairs in the MCh TLV.
- LastSVIDWants: A local temporary copy of the AdminSVIDWants.
- LocalSVIDPool: The set of SVIDs and bridge ports available for MCh assignment. These are determined by both administrative resource assignments and by resource availability. The OperSVIDList for a B role must be drawn from the LocalSVIDPool.
- LastLocalSVIDPool: A temporary copy of the LocalSVIDPool.

- OperVersion: The current value for the MCh version parameter. This value is included as the MCh version parameter in the local MCh TLV. The value VER0 = 100b means this version. The value DISABLE = 000b mean don't run MultiChannel.
- OperChnCap: The current value for the Number of Channels supported parameter. This value is included as the number of channels supported parameter in the local MCh TLV. The range for this variable is 1-0xffe.
- OperSVIDList: The current value for (channel,SVID) assignments. This is the list of (Channel,SVID) pairs included as the (Channel,SVID) pairs in the local MCh TLV. The total size of the list may not exceed 167 pairs. If the list is empty its value is NONE. The valid range for each channel of this list is from 1-0xffe. The valid range for each SVID in the list is from 1 to 0xfff. When the SVID is value is 0xfff the SVID is unconfigured. For the S role a SVID of 0xfff indicates a request for a channel. For the B role an SVID of 0xfff indicates an unconfigured channel.

- RemoteVersion: The current value for the remote MCh version parameter. This value is included as the Version parameter in the remote MCh TLV. NULL means no remote MCh TLV exists in the local LLDP database. The value for this variable may be VER0=100b setting any other value will result in stopping MultiChannel operation.
- RemoteChnCap: The current value for the Number of Channels supported parameter. This value is included as the number of channels supported parameter in the remote MCh TLV. NULL means no remote MCh TLV exists in the local LLDP database. The range for this variable is 1-0xffe.

- RemoteSVIDList: The current value for (channel,SVID) assignments. This is the list of (Channel,SVID) pairs included as the (Channel,SVID) pairs in the remote MCh TLV. NULL means no remote MCh TLV exists in the local LLDP database. If the list is empty but the MCh TLV is present its value is NONE. The total size of the list may not exceed 167 pairs. The valid range for each channel of this list is from 1-0xffe. The valid range for each SVID in the list is from 1 to 0xfff. When the SVID is value is 0xfff the SVID is unconfigured. For the S role a SVID of 0xfff indicates a request for a channel. For the B role an SVID of 0xfff indicates an unconfigured channel.

5.2.4 MDP Configuration Procedures

The MDP state machine uses three procedures. These are the SetSVIDRequest() procedure which is used place a new request from the station or set the initial TLV for a Bridge. The RxSVIDConfig() procedure is used by the station to configures a new set of S-VLANs and SVID assignments. The TxSVIDConfig() is used by the Bridge to respond to the station's request for S-VLANs.

- SetSVIDRequest(OperRole, AdminSVIDWants, OperSVIDList)
  - This function creates the OperSVIDList placed in the Local TLV database.
  - If the OperRole for the equipment is R then the OperSVIDList remains unchanged.
  - If the OperRole for the equipement is S two possible cases exist. In the first case we don't have any configured channels, indicated by OperSVIDList being equal to NONE. In this case the function places the AdminSVIDWants in OperSVIDList. In the second case we already have a running configuration indicated by the OperSVIDList not equal to NONE. In this case the function compares the AdminSVIDWants with the OperSVIDList. All active channels in the OperSVIDList which are in the AdminSVIDWants are kept active and in addition any channels not currently in the OperSVIDList are requested by including them in the OperSVIDList along with a 0xfff SVID number.
- RxSVIDConfig (OperSVIDs, LastRemoteVIDList)
  - This function creates the OperSVIDList placed in the Local TLV database for an S role equipment
  - The function compares the AdminSVIDWants with the LastRemoteSVIDList. For each AdminSVIDWants channel with an SVID assignment in the LastRemoteSVIDList a (Channel,SVID) pair is generated in the OperSVIDList. For each AdminSVIDWants channel without an SVID assignment in the LastRemoteSVIDList a (Channel,0xfff) pair is generated in the OperSVIDList.
- TxSVIDConfigB(OperChnCap, RemoteChnCap, LastLocalSVIDPool, RemoteSVIDList, OperSVIDList)
  - This function creates the OperSVIDList placed in the Local TLV database for an S role equipment
  - First the function takes the smaller of the OperChnCap and RemoteChnCap and truncates the RemoteSVIDList to the smaller of the two.
  - A new OperSVIDList is created as follows:
    - For each channel in the RemoteSVIDList with a (channel,SVID) pair in the OperSVIDList the (channel,SVID) remains unchanged unless the SVID is no longer part of the LastLocalSVIDPool. If the SVID is no long in the pool a new one is selected if available. If no SVID is

available the (channel,SVID) pair will be deleted from the OperSVIDList.

- For each channel in the RemoteSVIDList without a (channel,SVID) pair in the OperSVIDList an SVID is obtained from the LastLocalSVIDPool (the pool for Bridge resources) if available. If no SVID is available the (channel,SVID) pair will be deleted from the OperSVIDList.

5.2.5 MDP Configuration State Machines

The MDP state machine operates on TLV exchanged using LLDP operating at the LAN level (figure 28).

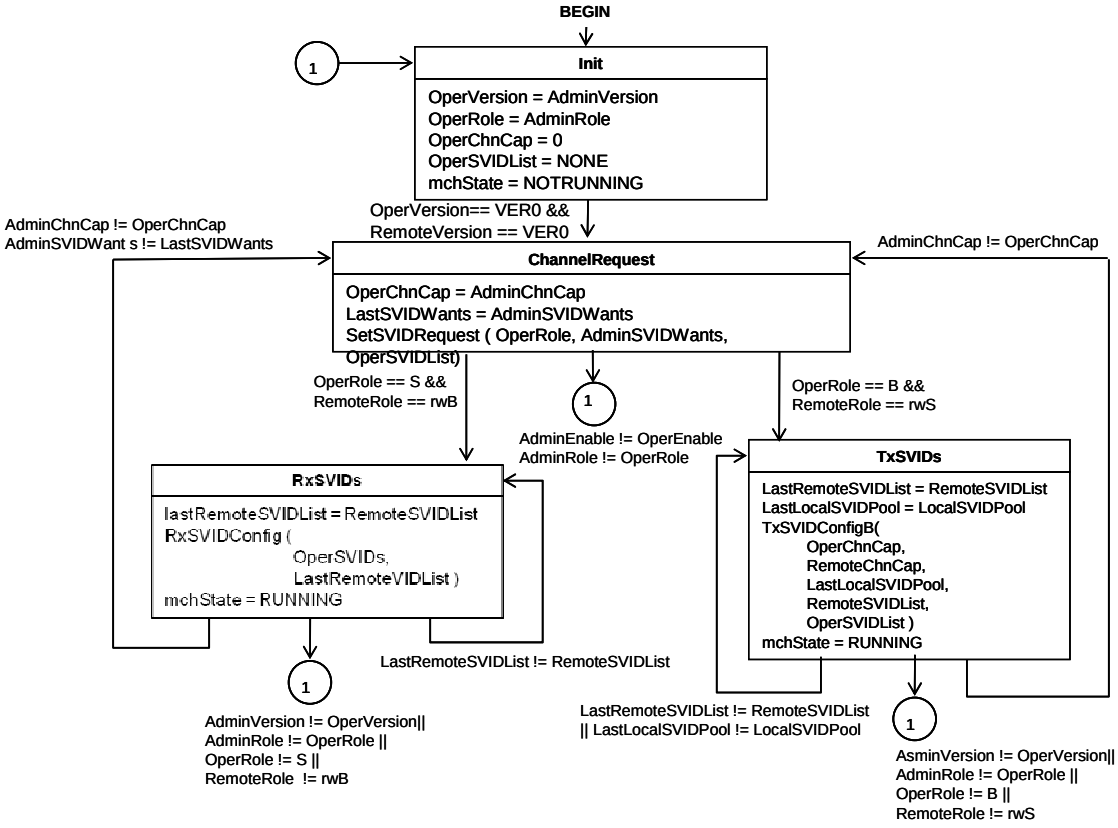


Figure 28. MDP State Machine

This LLDP instance is one per physical LAN associated with the Provider Network Port of the S-Component, which faces the LAN connecting the station to the bridge. If either the station or Bridge is not capable of multichannel operation no MCh-TLV will be inserted in the LLDP database. The absence of a MCh-TLV therefore indicates that the station or Bridge is not capable of multichannel. For MDP to progress both sides must indicate they are capable of MultiChannel operation, have the same version number and one side must have the 'B' role while the other side must have the 'S' role as indicated by the role bits of the MCh-TLV.

If both sides are MultiChannel capable, exactly one side has the 'S' role and one side has the 'B' role, and the 'B' has at least some of the resources requested by the 'S' side, the state machine will configure MultiChannel. The configuration proceeds by the 'B' providing the best match it can to the 'S's requested channels and configuration. The 'S' makes the resource request, the 'B' responds with its best matching resources, the 'S' then goes operational and reports its running configuration to the 'B', and finally the 'B' goes operational with the running configuration of the 'S'.

In the event the 'S' wishes to change its configuration it alters the request in its MCh-TLV and then follows the same process as above. If the 'B' losses its ability to support the current configuration it can alter the current configuration in its MCh-TLV at which time the 'S' must drop down to the resources supplied by the 'B'.

In the event of a change of the administered parameters the current operating S-VLANs must be terminated the configuration machine re-initialized.

6. Edge TLV Transport Protocol (ETTP) TLV and State Machine

This chapter will describe an architectural overview of the Edge TLV Transport Protocol (ETTP) protocol, followed by the ETTP TLV semantics and associated state machines.

6.1 Requirements

ETTP was designed with the following protocol requirements:

- Semantics associated with the <ULP, ETTP> interface:
  - A single link operating in "multi-channel" mode has one ETTP per channel.
  - For VSI, there is one VSI agent per channel ETTP and that agent may have multiple VSI instances sharing a single channel.
  - The <ULP, ETTP> interface is based on a complete ULPDU (i.e. the group of TLVs that are handed to ETTP for transmission).
    - The number of octets in the ULPDU may be less than the maximum number of octets that can fit into a ETTP frame.
    - The number of ULP TLVs may be less than the maximum number that can fit into a ETTP frame.
    - Procedures are used to describe how the ULP hands off ULP PDUs to ETTP and how ETTP hands off ULP PDUs to the ULP.
    - Given the <ULP, ETTP> interface is based on full ULPDUs, no immediate processing is needed at the ETTP level.
    - Outside the scope of this proposal are the semantics for handling: link down; and how multiple ULPs arbitrate when sharing the same ETTP.

- ETTP acknowledgement and sequencing semantics.
  - A ETTP acknowledge means the ETTPDU was received and there is a free buffer available to enable another send.
    - It doesn't mean the ETTPDU were delivered to the ULP.
  - At the transmit side, if a ETTP Acknowledge is not received within an Ack timer period, ETTP will retry the ETTPDU up to an EVB negotiated Retry Count is reached.
    - The value of the EVB negotiated Retry Count is on a per link basis not channel.
  - Once the receive side ETTP delivers the ULP PDU to the ULP through the receive side hand-off procedure, the ETTP buffer becomes available for another send.
    - The Acknowledgement must be sent in a separate ETTPDU (vs piggy backing onto a Transmit message in the opposite direction).
    - The receive side will issue a ETTP Acknowledgement after the completing the receive side hand-off procedure.
    - If the receive side hand-off procedure takes too long, the receive side ETTP may toss the ULP PDU and send back an ACK to indicate the ETTP buffer is free on the receive side.
      - Note: The length of time the send side waits, before tossing the ULPDU should not be less than the retransmission period times the maximum number of retries.
      - Semantics associated with slow ULP Data Unit reception (e.g. raising a flag) are outside the scope of this proposal.

- Sequence numbering must be used to detect duplicate vs new ETTPDUs.
- ETTP will not provide a keep-alive mechanism. Instead each ULP must do so.
- ETTP will not provide a digest at the ETTP level and any ULP Data Unit (or TLV) database synchronization is left up to the ULP.

Note, the intent is to have the server's virtualization infrastructure (e.g. Hypervisor) implement ETTP, versus having the NIC implement ETTP.

6.2 Edge TLV Transport Protocol Data Unit

This section specifies the format of a ETTP Data Unit, along with the header that is added to and removed from ETTP frames by the ETTP function. The ETTP header allows each ETTP Data Unit from the sender to be identified through a sequence number, which the receiver acknowledges by sending a ETTP Acknowledgement frame.

Ethertype = TBD	Sub-type	Mode	Sequence Number	ULPDU
← 2 Octets →	← 2 Octets →	← 1 Octet →	← 2 Octets →	Optional

Figure 29. ETTP Data Unit

The destination address of the Ethernet frame that contains a ETTPDU has the following semantics:

- Nearest bridge (01-80-C2-00-00-0E) for ETTP running at the link layer.
- Nearest Customer Bridge (01-80-C2-00-00-00) for ETTP running over a channel.
- Note, ETTP should also be allowed using a Uni-cast address.

The source address shall be the sending station or port individual MAC address.

A new Ethertype will be needed for ETTP. A ETTP exchange will run at the link if the link is not configured for multichannel. If ETTP is performed over Multi-channel, then the STAG for the channel shall precede the ETTPDU.

The ETTPDU contains:

- Sub-type - a 2 octet field that defines the ULP type included in the PDU. Note for Ack's the sub-type is ignored at the station.
- Mode – Identifies whether the operation is a:
  - ETTP request (0x00)
  - ETTP acknowledgement (0x01).
- Sequence number – identifies the sequential order of the PDU, with respect to other ETTPDUs. The starting sequence number may start anywhere for the first ETTPDU, but the sequence number for each subsequent new ETTPDU is incremented by 1.

6.3 ETTP Procedures

Two procedures are used to hand-off Data Units between the ULP and ETTP: ETTP_UNITDATA.request and ETTP_UNITDATA.indication. The implementation of these two procedures is outside the scope of this proposal. These <ULP, ETTP> interface procedures

may be implemented in many ways, including a queue. Also, the system must have a way of associating the ulptype with a specific ULP.

The ETTP_UNITDATA.request is invoked by the ULP at the sender to notify ETTP that a ULPDU is ready to be transmitted. The ulpdu parameter is a unit of work from the ULP. For example, for VSI it consists of a set of VSI TLVs passed from the VSI ULP to ETTP for transmission, where the set of TLVs must be less than or equal to the maximum allowed ETTPDU. Following is the format for the ETTP_UNITDATA.request procedure:

ETTP_UNITDATA.request (ulptype, ulpdu)

The ETTP_UNITDATA.indication is invoked by ETTP at the receiver to indicate a ULPDU has been successfully received and is available ULP processing. The ulpdu parameter is unit of work from the ULP. For example, for VSI it consists of a set of VSI TLVs passed from the ULP to ETTP for transmission, where the set of TLVs must be less than or equal to the maximum allowed ETTPDU. Following is the format for the ETTP_UNITDATA.indication procedure:

ETTP_UNITDATA.indication (ulptype, ulpdu)

6.4 ETTP State Machines

There are two state machines used by each ETTP instance: transmit and receive. The transmit state machine is invoked through the ETTP_UNITDATA.request procedure. The receive state machine is invoked upon reception of a ETTP Data Unit and it invokes the ETTP_UNITDATA.indication procedure.

6.4.1 ETTP Transmit State Machine

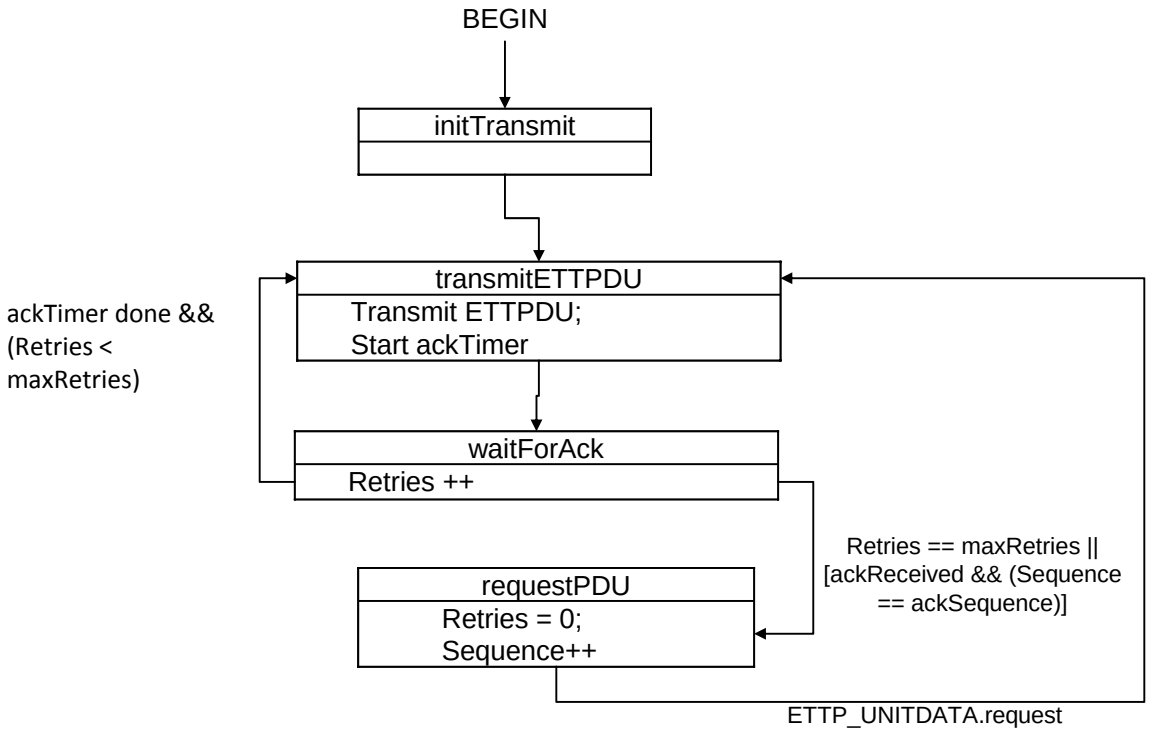


Figure 30. ETTP Transmit State Machine

The first entrance into transmitETTPDU is used to initiate the sequence counting on the receive side. That is, an ETTP Frame that simply contains the ETTP header is sent and an ackTimer is started. The waitForAck state waits for the L-ACK to be received that matches the last transmitted ETTP sequence number. If an L-ACK is received that matches the last transmitted ETTP sequence number or the number of retries exceeds the maximum number of retries, the sender will stop transmitting the ETTP Frame and proceed to requestPDU. If an L-ACK is not received within an ackTimer period and the number of retries is less than the maximum number of retries, the sender will retransmit the ETTP Frame. The requestPDU state increments the sequence count and waits for the ETTP_UNITDATA.request procedure to be invoked.

Note, the starting sequence number may start anywhere for the first ETTPDU. Also a Link Down event may restart the sequence number at the same point every time or not.

6.4.2 ETTP Receive State Machine

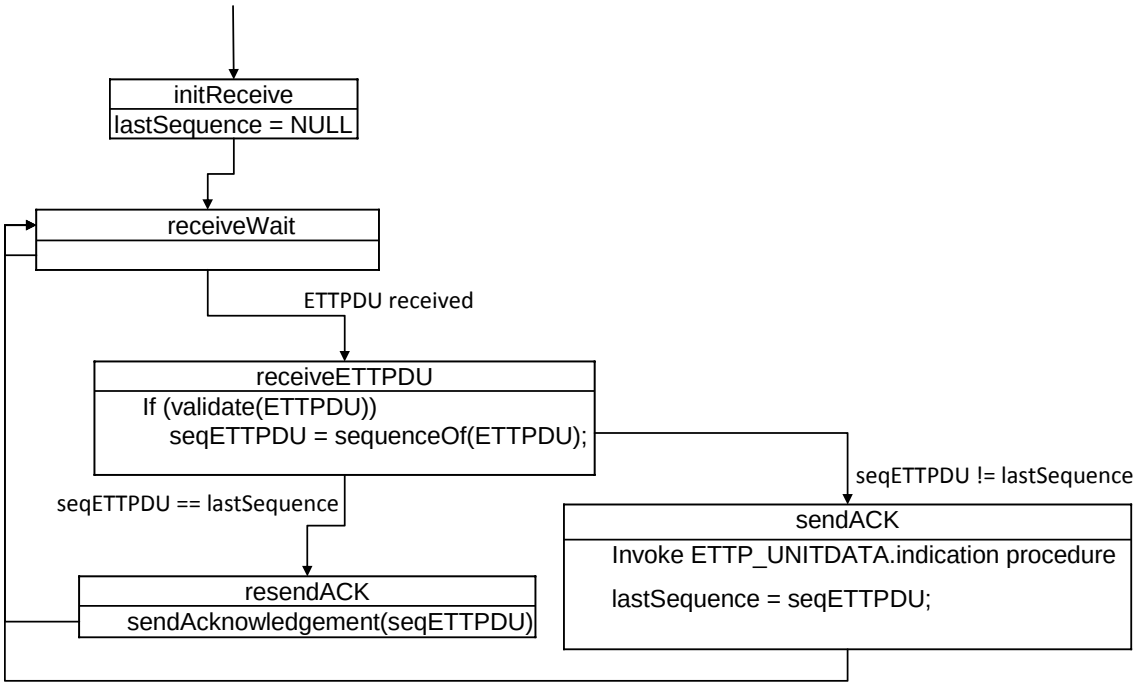


Figure 31. ETTP Receive State Machine

The first entrance into InitReceive is used to set the sequence counting to NULL and then proceed to receiveWait, which waits for an ETTP Data Unit to be received. The receiveETTPDU validates the ETTPDU and sets the current sequence number to the sequence number of the transmitted ETTPDU. If the current sequence number doesn't match the last transmitted ETTP sequence number, then in sendACK the ETTP Data Unit is delivered to the ULP and the lastSequence number is set to the current sequence number. If the current sequence number doesn't match the last transmitted ETTP sequence number, then in resendACK an L-ACK is sent.

7. Virtual Station Interface (VSI) TLV and State Machine

This section covers the Virtual Station Interface (VSI) Discovery and Configuration Protocol (VDP) and State Machine. VDP uses ETTP (Enhanced TLV Transport Protocol) for VDP exchanges.

7.1.1 VSI Discovery and Configuration TLV

VSI TLV is used for discovery and configuration and is exchanged between the Station and Bridge. One or more VSI TLVs are transported in an ETTP Data Unit. Following is the format and semantics for a VSI TLV:

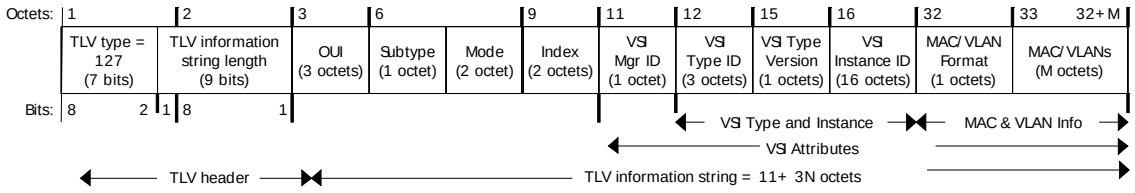


Figure 32. VDP TLV

Index – VSI index – Offset in bit-arrays containing state and configuration status of VSIs.

Mode – Indicates VSI TLV Mode

- First octet identifies a pre-associate, associate, de-associate, or the corresponding confirmation or rejection for each.
- Second octet is used during a rejection to indicate the reason for the pre-assoc or assoc rejection.

VSI Manager ID – Identifies the VSI Manager with the Database that holds the detailed VSI type and or instance definitions. VSI Manager ID can be used to obtain IP address and/or other connectivity and access information for the manager.

VSI Type ID (VTID) – The integer identifier of the VSI Type.

VSI Type ID Version – The integer identifier designating the expected/desired version of the VTID

VSI Instance ID – A globally unique ID for the connection instance. The ID shall be done consistent with IETF RFC 4122.

Format – identifies the format of the MAC and VLAN information that follows in the TLV. Note, the VSI TLV allows multiple formats, which makes possible extensions in the future.

MAC/VLANs – Listing of the MAC/VLANs associated with the Virtual Station Instance (VSI).

Following is the format for Format = 1

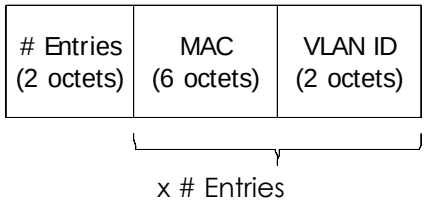


Figure 33. VDP Format = 1 Schema

Note, the station and switch environments and their common understanding of the VTID meaning is outside the scope of this TLV. Also, the contents of a VSI Type are outside the scope of this proposal.

7.1.1.1 VSI TLV – Mode and Mode Response

The purpose of the Mode field is to identify the type of VSI TLV. It is defined as follows.

VSI TLV Request field: 1st octet

Pre-Associate:	0x00
Pre-Associate with resource reservation:	0x01
Associate:	0x02
De-Associate:	0x03

VSI TLV Response field: 2nd octet

For all the responses, the bridge reflects the same VSI TLV fields as the Requester had sent. On requests, response field is initialized to 0x00 (Success). Following are the possible values of the response field.

Success: 0x00
The VSI Request was successfully completed by the switch

Invalid Format: 0x01
The VSI Format is not supported by the switch

Insufficient Resources: 0x02
The switch does not have enough resources to complete the VSI operation successfully.

Unused VTID: 0x03
The VSI referenced by the VSIID does not exist in the VSI Manager database referenced by the VSI Manager Identifier

VTID Violation: 0x04
The VSI referenced by the VSIID is not allowed to be associated with the VTID.

VTID Version Violation: 0x05
The VSI referenced by the VSIID is not allowed to be associated with the VTID Version.

Out of Sync: 0x06
The VTID or one of the VSI List fields used in the Associate is not the same as the corresponding field used in the Pre-Associate.

Reserved 0x08 – 0xFF

These Responses are reserved for future use.

Mode and Mode Response fields are used under the control of VDP state machines.

7.1.1.2 VSI TLV Mode and Responses Semantics

Following are the semantics association with each VSI TLV Request.

7.1.1.2.1 Pre-Associate

The Pre-Associate is used to pre-associate a VSI Instance Identifier to a VSI Type ID. If required, the bridge should obtain VSI Type Definition from the VSI Manager Database. The bridge must validate the request (see below) and fail it in case of errors (see below for responses). Successful Pre-Association does not enable any traffic from VSI. Note that VSI may still be associated at another station. The Pre-Associate enables faster response to an Associate, by allowing the bridge to obtain VSI Type state, prior to an association.

The second Mode octet is used by the bridge to communicate the results of the Pre-Associate requested for the VSI Instance ID (VSIID).

Following are the mode and responses with their semantics:

- Success - Pre-Associate was successful. The switch shall permit a subsequent Associate or De-Associate by the VSI referenced by the VSI Instance Identifier.
- The following are all unsuccessful Pre-Associate Completions. For each of these, the switch shall not permit a subsequent Associate or De-Associate by the VSI referenced by the VSIID.
  - Invalid Format.
  - Insufficient PT Resources.
  - Unused VTID
  - VTID Violation
  - VTID Version Violation

Pre-Associate requires resource lease timer mechanism to conserve Bridge resources. Pre-Associate does not allow any traffic from VSI which is enabled when the VSI is Associated.

7.1.1.2.2 Pre-Associate with Resource Reservation

Pre-Associate with Resource Reservation has same steps as Pre-Associate but also reserves resources.

Bridge should validate required resources and place reservation to ensure resources for subsequent Associate step. Pre-Associate requires resource lease timer mechanism to conserve Bridge resources. Pre-Associate does not allow any traffic from VSI which is enabled when the VSI is Associated.

Second Mode octet contains the results of the Pre-Associate requested for the VSI Instance ID (VSIID). Following are the mode and responses with their semantics.

- Success - Pre-Associate with Resource Reservation was successful. The switch shall permit a subsequent Associate or De-Associate by the VSI referenced by the VSI Instance Identifier.
- The following are all unsuccessful Pre-Associate with Resource Reservation Completions. For each of these, the switch shall not permit a subsequent Associate or De-Associate by the VSI referenced by the VSIID.
  - Invalid Format.
  - Insufficient PT Resources.
  - Unused VTID
  - VTID Violation
  - VTID Version Violation

7.1.1.2.3 Associate

Associates the VSI Instance ID with the VSI Type ID (VTID). If VSI Type definition is not already cached in the bridge, the bridge fetches the VSI Type definition from the VSI Type definition Database. Bridge allocates required bridge resources for the referenced VSI. The Bridge binds specific MAC/VLAN pairs with the VSI Type ID which allows classification of L2 traffic to the VSI and enforcing of VSI Type controls. Bridge activates the configuration for the VSI Type ID. This association is then applied to the traffic flow from/to the VSI Instance.

For a given VSI Instance ID, a Station may issue an Associate without having previously issued a Pre-Associate or Pre-Associate with Resource Reservation. Same VSI Instance may not be successfully Associated more than once on two different bridges or ports.

In VSI TLV, second octet in the mode field contains the results of the Associate request performed for the VSI Instance Identifier. These are described below.

- Success - Associate was successful. Prior to issuing this response, for a format 1 VSI TLV, the bridge shall associate the VSI Type referenced by the VSI Type Identifier and VSI Type Version with the MAC Address, VLAN and VSIID.
- The following are all unsuccessful Associate Completions.
  - Invalid Format
  - Insufficient Resources - If the Associate was preceded by a successful Pre-Associate with Resource Reservation, then the bridge shall not issue this response.
  - VTID Violation
  - VTID Version Violation
  - Out of Sync

7.1.1.2.4 De-Associate

De-associate a VSI Instance Identifier from the associated VTID. Pre-Associated and Associated VSIs can be De-Associated. De-Associate releases resources and de-activates the configuration associated with the VSI instance. A VSI Instance may get De-Associated by bridge due to bridge error situation or management action.

In VSI TLV, second octet in the mode field contains the results of the De-Associate request performed for the VSI Instance Identifier. These are described below.

- Success - De-Associate was successful. Prior to issuing this response, for a format 1 VSI TLV, the bridge shall de-associate the VSI Type referenced by the VSI Type Identifier from the the MAC Address, VLAN and VSI Instance ID.
- The following are all unsuccessful De-Associate Completions.

    - Invalid Format
    - VTID Violation
    - VTID Version Violation

Note: The result of the above semantics is that De-Associate can be issued at any time.

7.1.1.2.5 VSI Type ID (VTID) Semantics

VSI Type ID (VTID) is an integer value field used to identify a pre-configured set of controls/attributes that are to be associated with a set of VSIs.

VTID contents and meaning and the database used to contain the VSI Type are outside the scope of this effort. One VTID may describe the VSI Type configuration of multiple VSIs. The VSI Type content referenced by the same VTID may differ between switches and VEBs. For example: same VTID is used by switches from two different vendors; or same VTID is used by a VEB and vendor switches.

7.1.1.3 VSI Type ID Version Semantics

VTID Version is integer identifier designating the expected/desired VTID version.
The VTID Version enables a VSI Manager Database to contain multiple VSI Type versions. It allows smooth migration to newer VSI types.

7.1.1.4 VSI Instance ID

VSI Instance ID is a globally unique ID for the VSI instance. The ID shall be done consistent with IETF RFC 4122. VSI ID is gets generated when VSI instance is created by VSI Instance Manager at request of VM Manager. VSI Instance creation mechanism is outside scope of this proposal but expected to be created by VM Manager or VSI Manager.

7.1.1.5 MAC – VLAN Information Format

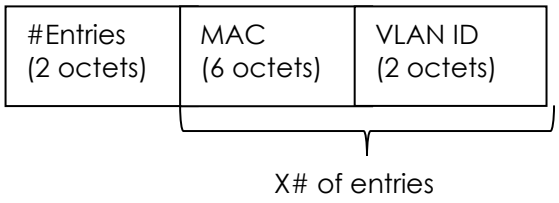


Figure 34. MAC-VLAN Information Format 1

MAC-VLAN Format-1 contains the set of MAC Addresses and VLANs to be associated with the VSI Instance ID. Note the bridge uses MAC+VID to identify traffic from VSI and to steer the frames.

Field:

#MAC-VLAN pairs:	2 octets

Per MAC-VLAN Pair Content:

MAC address:	48 bits
VID:	12 bits

7.1.2 VDP Requirements and Assumptions

Following are VDP requirements associated met by VDP state machines described in this section:

1. VDP must support a VSI Pre-Associate (with and without resource reservations), Associate and De-Associate.
2. Associate, Pre-Associate and De-Associate are Idempotent i.e. can be repeated.
3. The bridge must allow for an Associate to be issued without the need for a previous Pre-Associate.
4. VDP may be used in conjunction with both a VEPA and VEB.
5. VDP utilize ETTP as the transport for a VDP Data Unit that contains one or more VDP TLVs. VDP utilizes the following capabilities of ETTP:
    1. Transport will be transmitting TLVs in-order and are received in-order.
    2. Flow control
    3. Transport error from ETTP and LLDP are indicated to VDP
    4. ETTP provides best effort delivery of TLV. At the Station, if a VDP Acknowledgement is not received, within an Acknowledgement timeout period, VSI exits the state machine. The Acknowledgement timeout period is defined as 2*ETTP retransmission period * Maximum number of retries, plus a locally administered wait that is outside the scope of this document.
6. Health TLV mechanism to ensure:
    1. Bridge resources are not reserved for too long a time period for inactive VSIs (lease semantics)
    2. Allow removing resources from inactive VSIs with the goal of
        1. Conserving bridges resources (Number VSIs being handled by bridge can be large).
        2. Prevent inactive or VMs in error state to continue to hold resources.
    3. For multichannel, timeout out values to be negotiated on a per channel basis between station and bridge. One timeout used for all ULPs on the channel negotiated using EVB TLV.
    4. If multichannel is not enabled, timeout out values to be negotiated per link basis between station and bridge. One timeout used for all ULPs on the link negotiated using EVB TLV.
7. Ensure VSI state and configuration between the Station and the Bridge remains consistent.
8. Hard errors at the Bridge or the Hypervisor that can impact individual VSI or Hypervisor/Bridge as a whole are handled by removing all VSI configuration.

9. Bridge and Station Errors are detected through one or more of the following mechanisms.
    1. VSI KEEP-ALIVE (periodic transmission of VSI TLV from station and response from Bridge)
    2. ACK Timer
10. Supports for switch/hypervisor administrator actions that force VSI De-Associate.
11. Should enable statistics and logging capability.

7.1.3 VDP – Local Variables and Procedures

vsiState: Local variable for current state.

localTLV: Current local (active) TLV (configuration)

AdminTLV: TLV from local administration. In addition appropriate localChange variable is set. It allows mode change

RemoteTLV: TLV received from remote.

TxTLV(*vsiTLV*): Transmits AdminTLV using TLV transport/DBA service interfaces

ProcRxAndSetCfg**(***vsiRemoteTLV*,*vsiLocalTLV*,*vsiState***):**
Processes receive TLV and Sets local TLV variable based on received Remote TLV and vsiState. In case of error, returns error. This function handles PreAssociate with and without resource reservation case as well as accessing VSI Type definition fetch, if required.

StartACKtimer(): Resets ACKTimeout local variable to FALSE and Starts ACK timer. Response (ACK or NACK is expected before timer expires.

ACKTimeout: This local variable is set to true, if ACK timer expires

vsiErrorPerm(*vsiRemoteTLV*):
processes the vsiRemoteTLV and returns TRUE is response code is unrecoverable (permanent) error.

The next sections contain the VSI State Machine. Following are notes regarding those state machines:

1. The purpose of the ACKtimer is to catch the unusual case of a TLV getting lost. The following architectural minimum shall be used: The Acknowledgement timeout period is defined as 2*ETTP retransmission period * Maximum number of retries, plus a locally administered wait that is outside the scope of this document.
2. For any VSI ACK received for a non-active VSI the station shall drop the packet.
3. VSI State is set to NULL on exit.
4. The VSI State Machine does will not implement retry mechanism on NACK. Instead the ULP can process the NACK reasons and retry the VSI operation.
5. VDP state machine will exit on receiving NACK.

7.1.4 Station VSI State Machine

Following is the VSI State Machine for the Station.

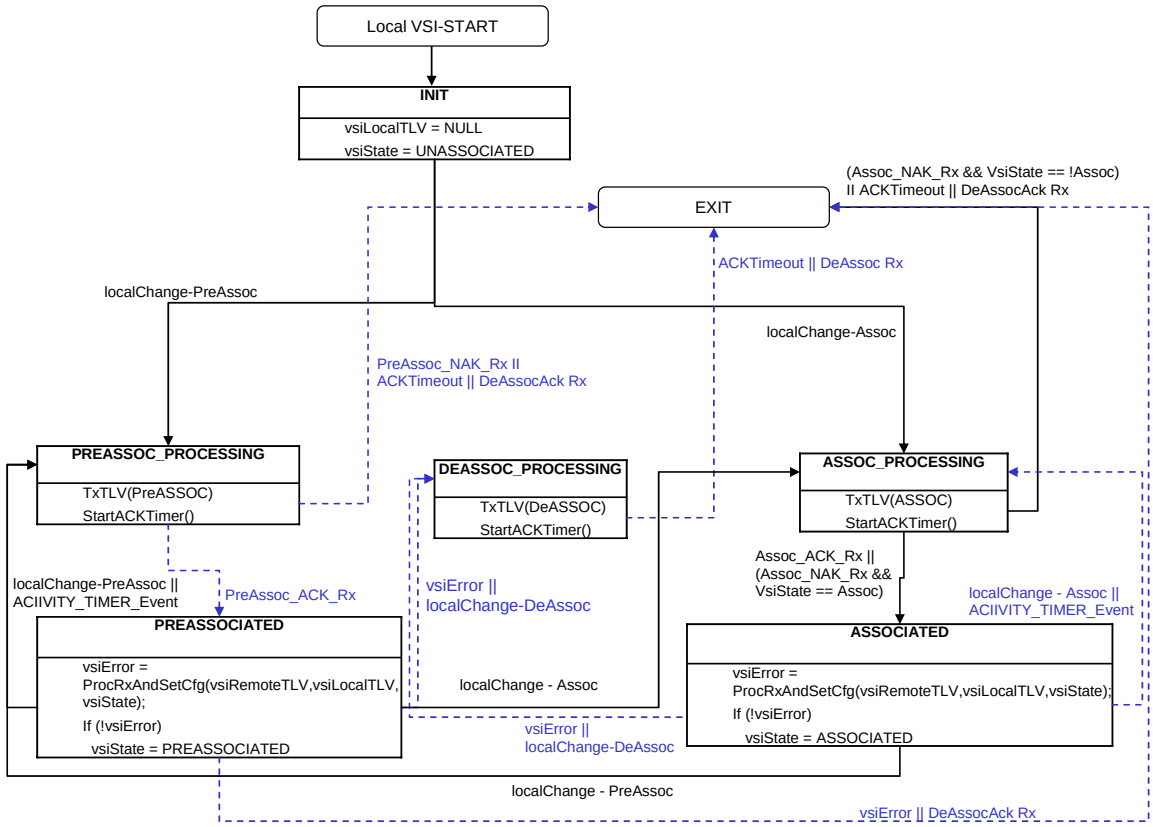


Figure 35. Station's VSI State Machine

7.1.5 Edge Bridge VSI State Machine

Following is the VSI State Machine for the Bridge.

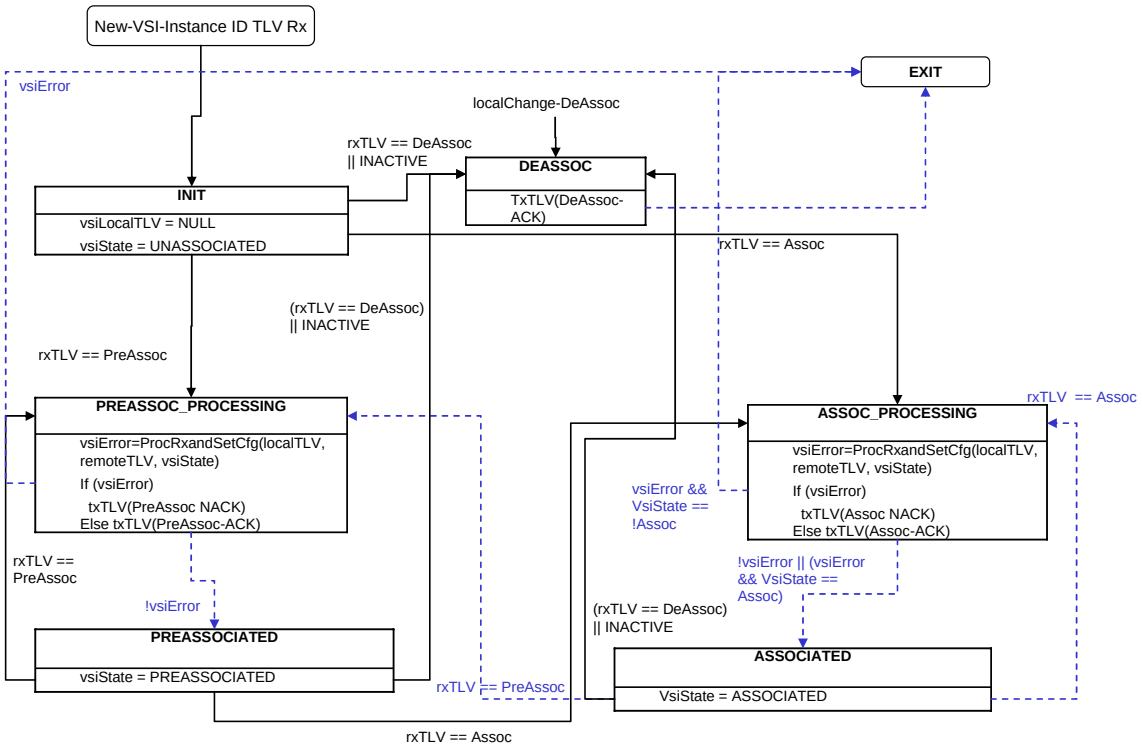


Figure 36. Edge Bridge's VSI State Machine

8. Glossary

Term	Description
Channel	An S-VLAN used to associate a set of VSI Instances with a physical Ethernet link. Traffic within one channel is isolated from traffic in another channel on the same link through the use of a S-Tag.
Chassis	A physical component incorporating one or more IEEE 802 LAN stations and their associated application functionality.
Chassis identifier	An administratively assigned name that identifies the particular chassis within the context of an administrative domain that comprise one or more networks.
CVID	Customer VLAN Identifier
DA	Destination Address
DS	Distribution System
Edge Virtual Bridging (EVB)	The environment where physical end stations, containing multiple VSI Instances, all require the services of adjacent bridges forming a LAN. EVB environments are unique in that virtual NIC configuration information is available to the EVB device that is not normally available to an 802.1Q bridge.
EUI	Extended Unique Identifier
Hypervisor	Computer software and / or hardware platform virtualization software that enables multiple operating systems to operate on top of common, shared hardware.
ID	Identifier
IEEE 802 LAN	Local area network (LAN) technologies that provide a media access control (MAC) Service equivalent to the MAC Service defined in ISO/IED 158001-1. IEEE 802 LANs include IEEE Sd. 802.3, IEEE Std 802.11, IEEE Std 802.16, IEEE Std 802.17, and ISO 9314-2 LANs.
IEEE 802 LAN Station	An IEEE 802-compatible entity that incorporates all the necessary mechanisms to participate in media access control of an IEEE 802 LAN, and that is at least capable of providing the MAC service plus the mandatory capabilities of the LLC.
LLC	Logical Link Control (sub-layer)
Link Layer Discovery Protocol (LLDP)	A media-independent protocol capable of running on all IEEE 802 LAN stations and to allow an LLDP agent to learn the connectivity and management information from adjacent stations.
LLDP agent	The protocol enttity that implements LLDP for a particular MSAP associated with a Port.
LLDPDU	Link Layer Discovery Protocol Data Unit
LSAP	Link Service Access Point
MAC	Media Access Control
MAC service access point (MSAP)	The access point for MAC services provided to the LLC sub-layer.
MSAP Identifier	The identifier of a MAC service access point.

Term	Description
Management entity	The protocol entity that implements a particular network management protocol and that provides access support to a MIB associated with the protocol and implemented in a host chassis.
Management Information Base (MIB)	The instantiation of all MIB modules in a managed entity (e.g. system or device)
Management Information Base module (MIB module)	The specification or schema for a data base that can be populated with information required to support a network management information system.
Multi-Channel	The capability to multiplex multiple virtual channels over a single physical Ethernet link.
Network	An interconnected group of systems, each comprising one or more IEEE 802 LAN stations.
Network Interface Controller (NIC)	A device that includes a non-forwarding IEEE 802 LAN station.
Network Management System (NMS)	A management system that is capable of utilizing the information in a MIB.
Object identifier (OID)	An identifier used to name an objective. Structurally, an OID consists of a node in a hierarchically-assigned namespace, formally defined in ISO/IEC 8824-1. Abstract Syntax Notation 1 (ASN.1). OIDs are used in this standard to identify MIB modules and the objects they contain.
OUI	Organizationally Unique Identifier
Physical network topology	The identification of systems, of IEEE 802 LAN stations that compose each system, and of the IEEE 802 LAN stations that attach to the same IEEE 802 LAN.
PCI	Peripheral Component Interface as defined by the PCI-SIG. http:www/pcisig.com. PCI Express (PCIe) represents the latest incarnation of PCI technology within the industry.
PD	Powered Device
Port	The entity in a chassis/system to support an MSAP. A port incorporates one and only one MSAP and identifies the collection of manageable entities that provide the MAC Service at the MSAP.
Port identifier	An administratively assigned name that identifies the particular port within the context of a system, where the identification is convenient, local to the system, and persistent for the system's use and management (whereas the MAC address that globally identifies the MSAP can not be).
PVID	Port VLAN ID
Reflective Relay	Frame relay where the destination port is also the source port
SA	Source Address
Service VLAN	A VLAN identified by a S-VID
Service VLAN ID (S-VID)	A VLAN identifier conveyed in an S-TAG
Service VLAN Tag (S-Tag)	A VLAN tag with a Tag Protocol Identification value allocated for "802.1Q Service Tag Type"

Term	Description
Single-Root I/O Virtualization (SR-IOV)	PCI-SIG specification that enables a PCIe Device to be simultaneously shared by multiple operating systems. A SR-IOV Device supports multiple PCI physical functions (PF) and virtual functions (VF). A PF or a VF is made visible to an operating system by a hypervisor as though it is a single, non-shared PCI Function.
S-VLAN component	A VLAN-aware bridge component with each Port supported by an instance of the IESS that can recognize, insert, and remove Service VLAN tags.
SVID	Service VLAN Identifier
System	A managed collection of hardware and software components incorporating one or more chassis, stations, and ports.
Type, length, value (TLV)	A short, variable length encoding of an information element consisting of sequential type, length, and value fields where the type field identifies the type of information, the length field indicates the length of the information field in octets, and the value field contains the information itself.
VID	VLAN ID
VDP	Virtual Station Interface Discovery and Configuration Protocol. The protocol used to discover and configure a Virtual Station Interface Instance.
Virtual Ethernet Bridge (VEB)	A VEB is a frame relay service that supports local bridging between multiple VSI Instances and (optionally) the external bridging environment. A VEB may be implemented in software as a vSwitch or as embedded hardware within a NIC.
Virtual Ethernet Port Aggregator (VEPA)	A Virtual Ethernet Port Aggregator (VEPA) is a capability within a physical end station that collaborates with an adjacent, external bridge to provide bridging support between multiple virtual end stations and external networks. The VEPA collaborates by forwarding all station-originated frames to the adjacent bridge for frame processing and frame relay (including reflective relay forwarding) and by steering and replicating frames received from the VEPA uplink to the appropriate destinations. May be implemented in software or in conjunction with embedded hardware. *Note: As with the case of VEBs, VEPAs have access to vNIC configuration information that normally is not available to an 802.1Q bridge.*
Virtual Machine (VM)	An operating system running on top of a hypervisor.
Virtual Port (vPort)	A vPort is a logical Port associated with one end of a channel. Within a physical end station, one or more VSI may be multiplexed on top of a vPort. Within an adjacent bridge, a vPort represents a virtual bridge port.
Virtual Station Interface (VSI)	A physical or software emulated end station connected to a VEB (vSwitch or embedded hardware within a NIC), a VEPA or directly to an S-VLAN Component.
Virtual Switch (vSwitch)	A software emulated bridge typically implemented within the server virtualization infrastructure (e.g. a Hypervisor). A vSwitch switches network packets between multiple operating systems executing on common, shared hardware. See also VEB.

Appendix – VDP Exchange Examples

8.1 VSI PreAssociate, Associate and DeAssociate

The following example depicts the VDP exchanges used to Pre-Associate, Associate and De-Associate a VSI Instance with a VSI Type, VSI Type Version and set of MAC Address and VLAN pairs.

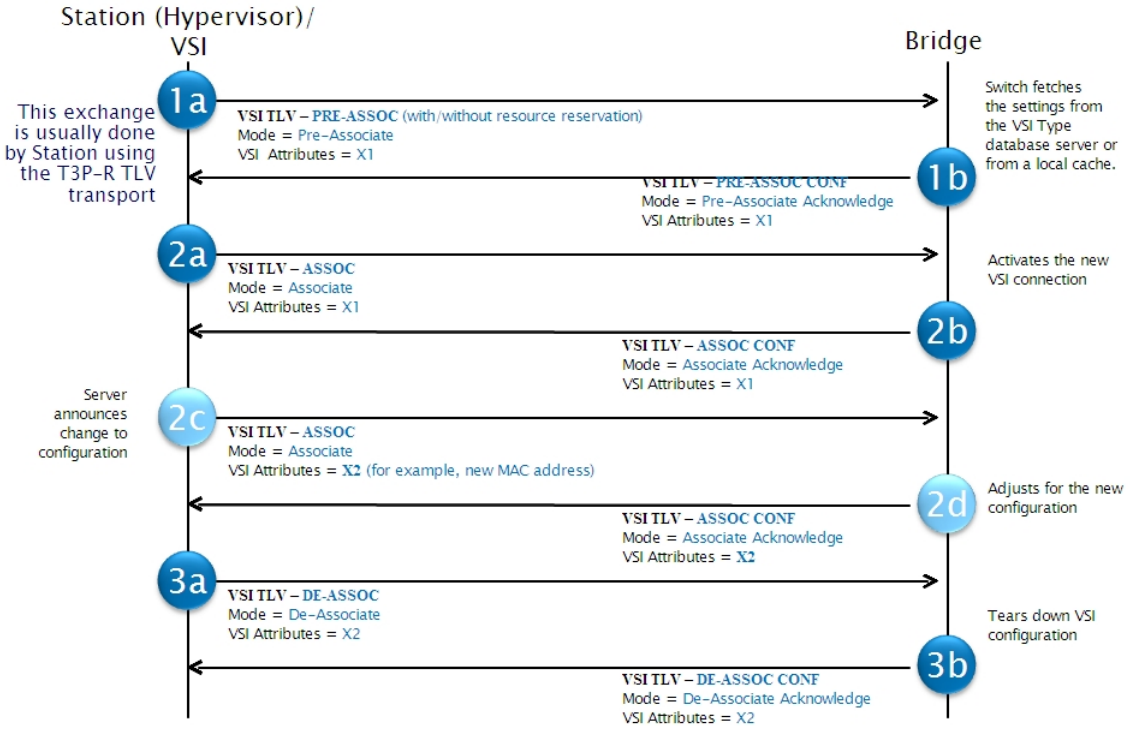


Figure 37. VSI PreAssociate, Associate and DeAssociate Exchange

8.2 VSI Transport Error Case

The following example depicts the VDP exchange associated with a lost EETP transmission of a VSI Associate Request Acknowledgement, showing ETTP retrying the transmission.

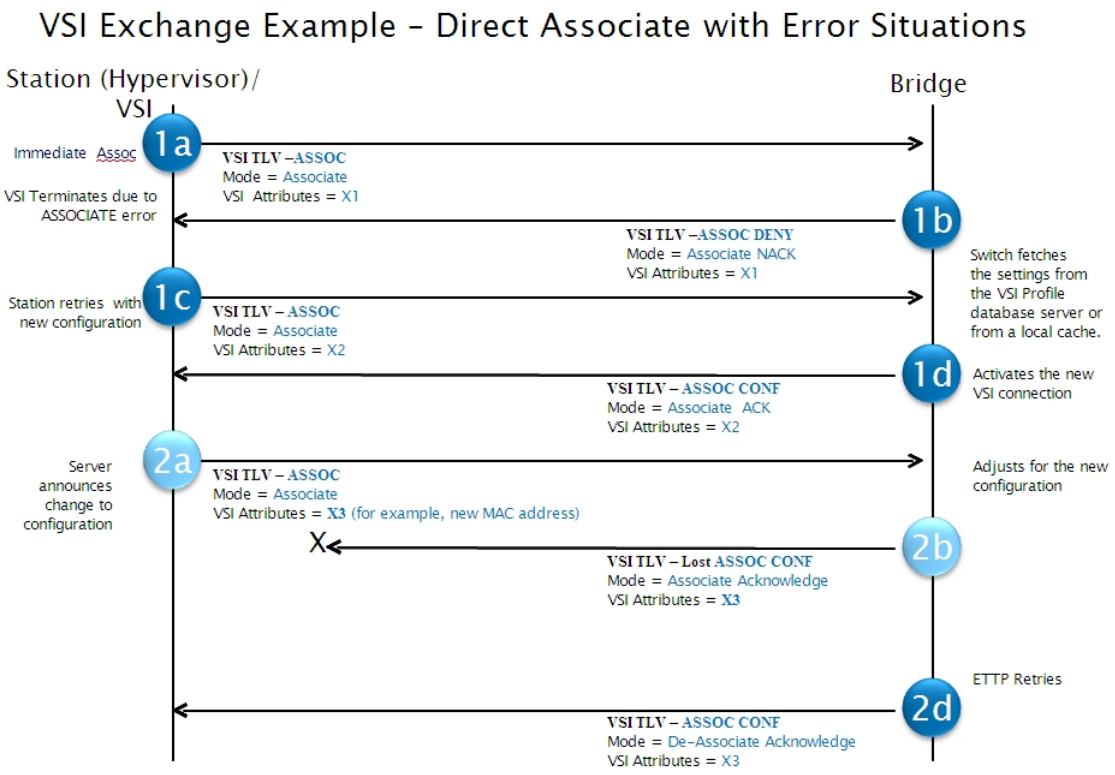


Figure 38. VSI Transport Error

8.3 VSI PreAssociate Resource Lease Refresh Exchange

The following example depicts the VDP exchange associated with an inactive VSI Instance in the Pre-Associated state, where the bridge's VSI State Machine forces a De-Association.

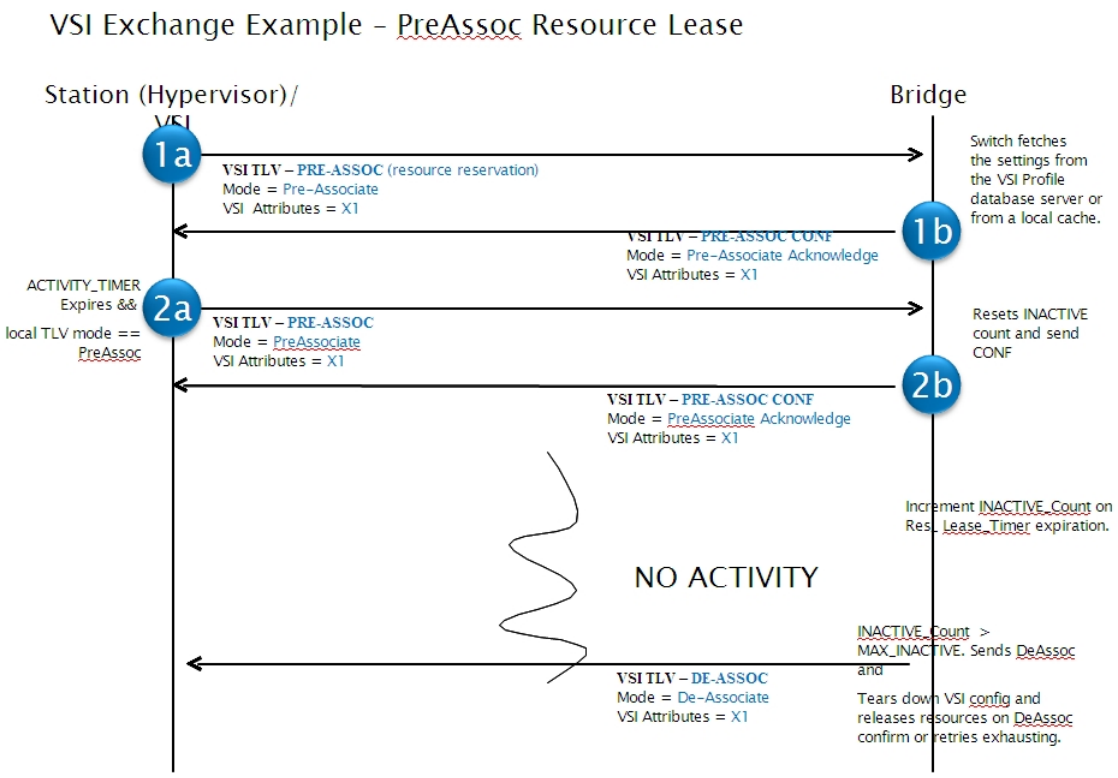


Figure 39. PreAssociate Resource Lease Exchance

8.4 VSI Associate Resource Lease Exchange

The following example depicts the VDP exchange used with an inactive VSI Instance in the Associated state, where the bridge's VSI State Machine forces a De-Association.

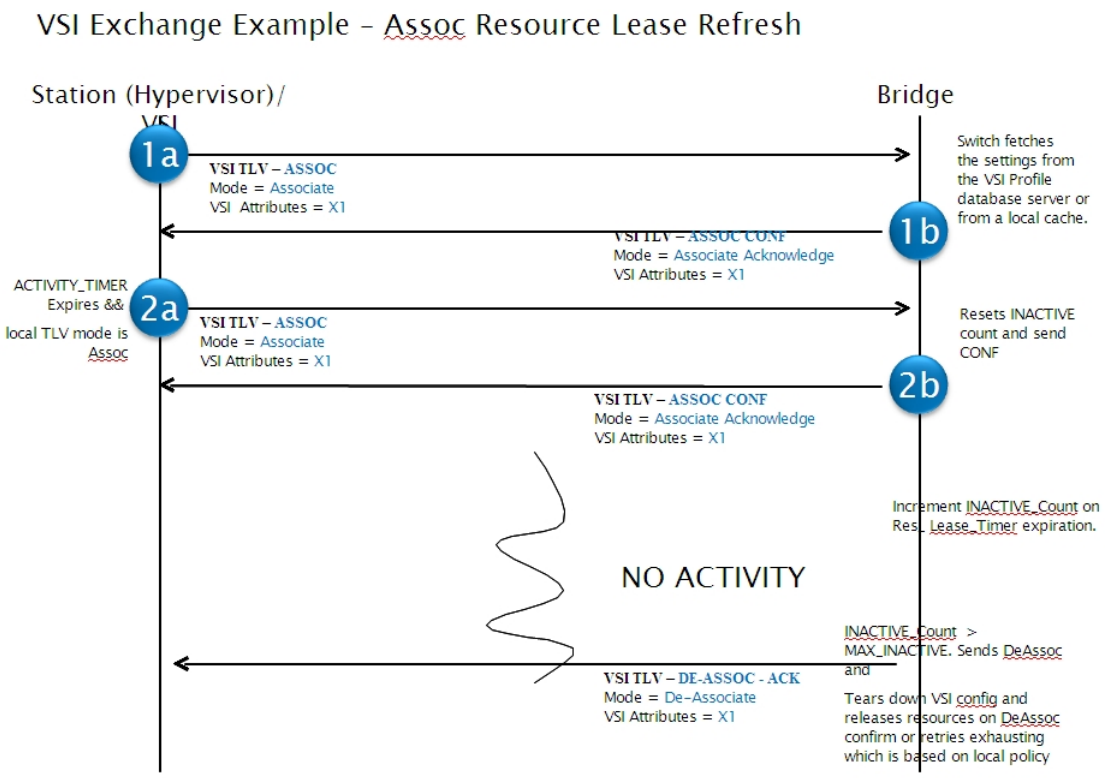


Figure 40. Associate Resource Lease Exchange