VNTag 101

Joe Pelissier
jopeliss@cisco.com

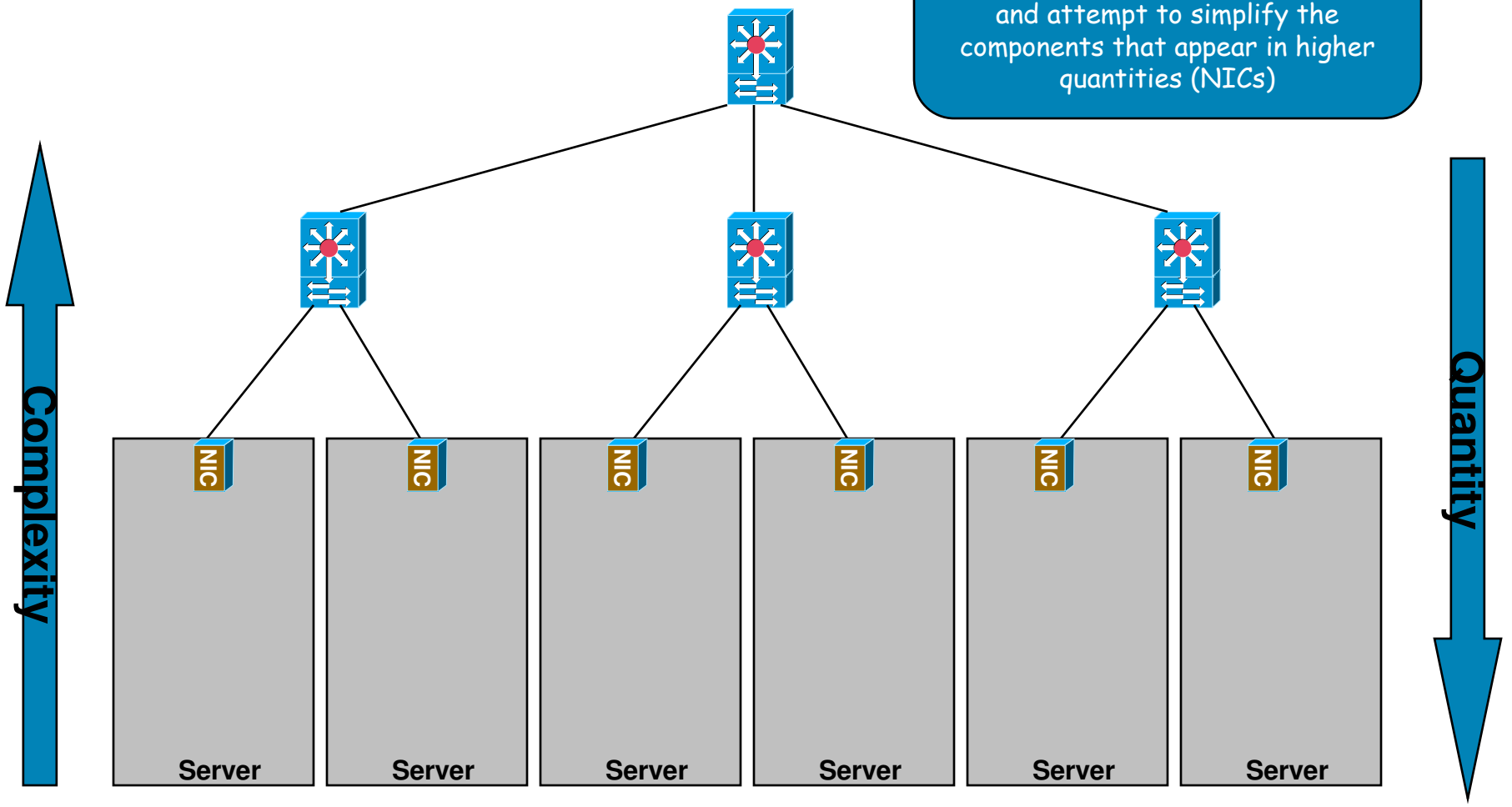
new-pelissier-vntag-seminar-0508

Agenda

- Motivation, Problem Statement, and Requirements
- An Approach
- The Path to Tagging
- The VNTag Proposal
- VNTag Addressing Examples
- Case Studies
- Coexistence of VEB and IV / VEPA
- A Call for Interest
- Summary

Motivation

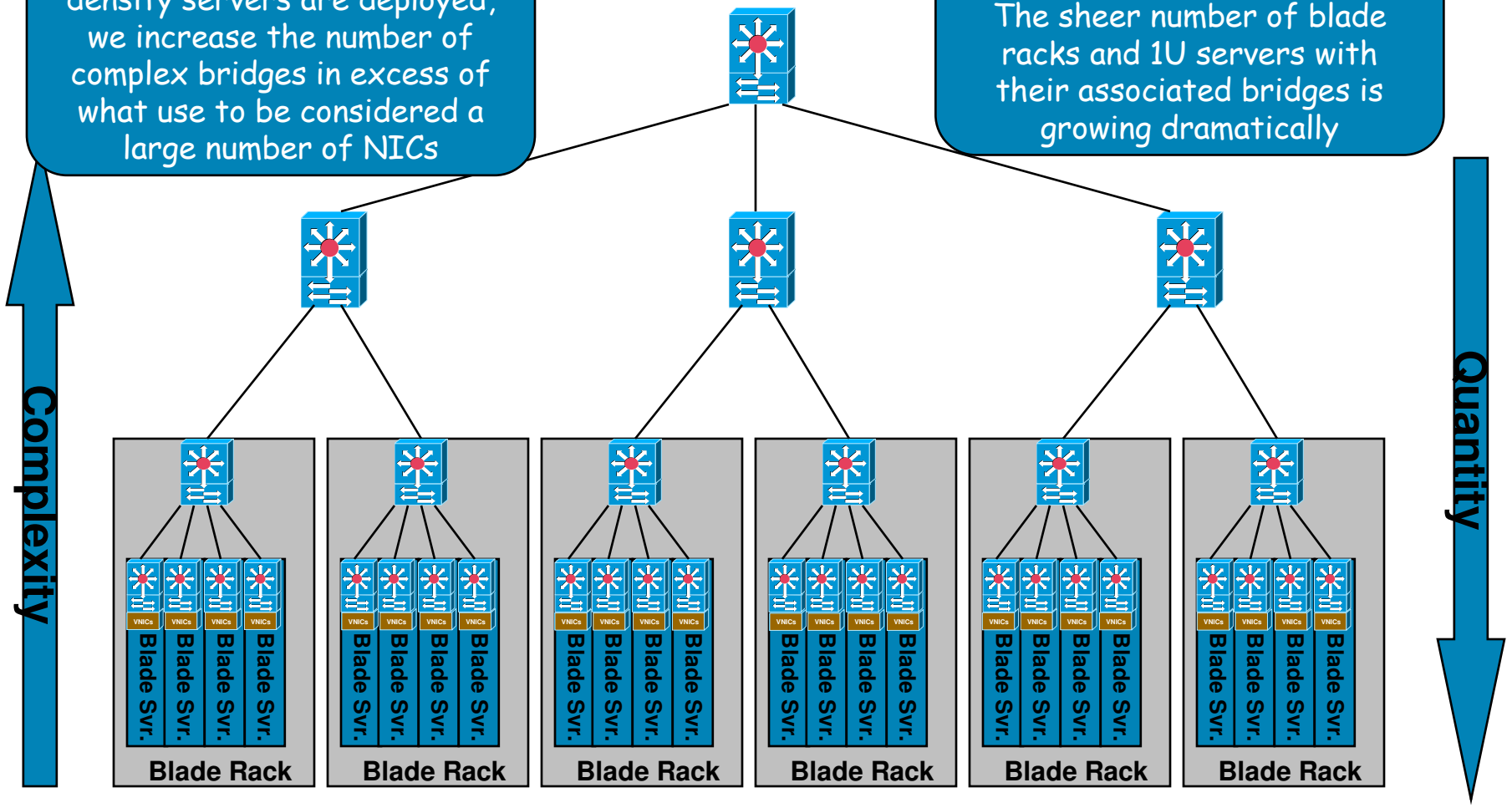
As a general rule, we push complexity up into the components of which we have fewer (bridges), and attempt to simplify the components that appear in higher quantities (NICs)



Motivation

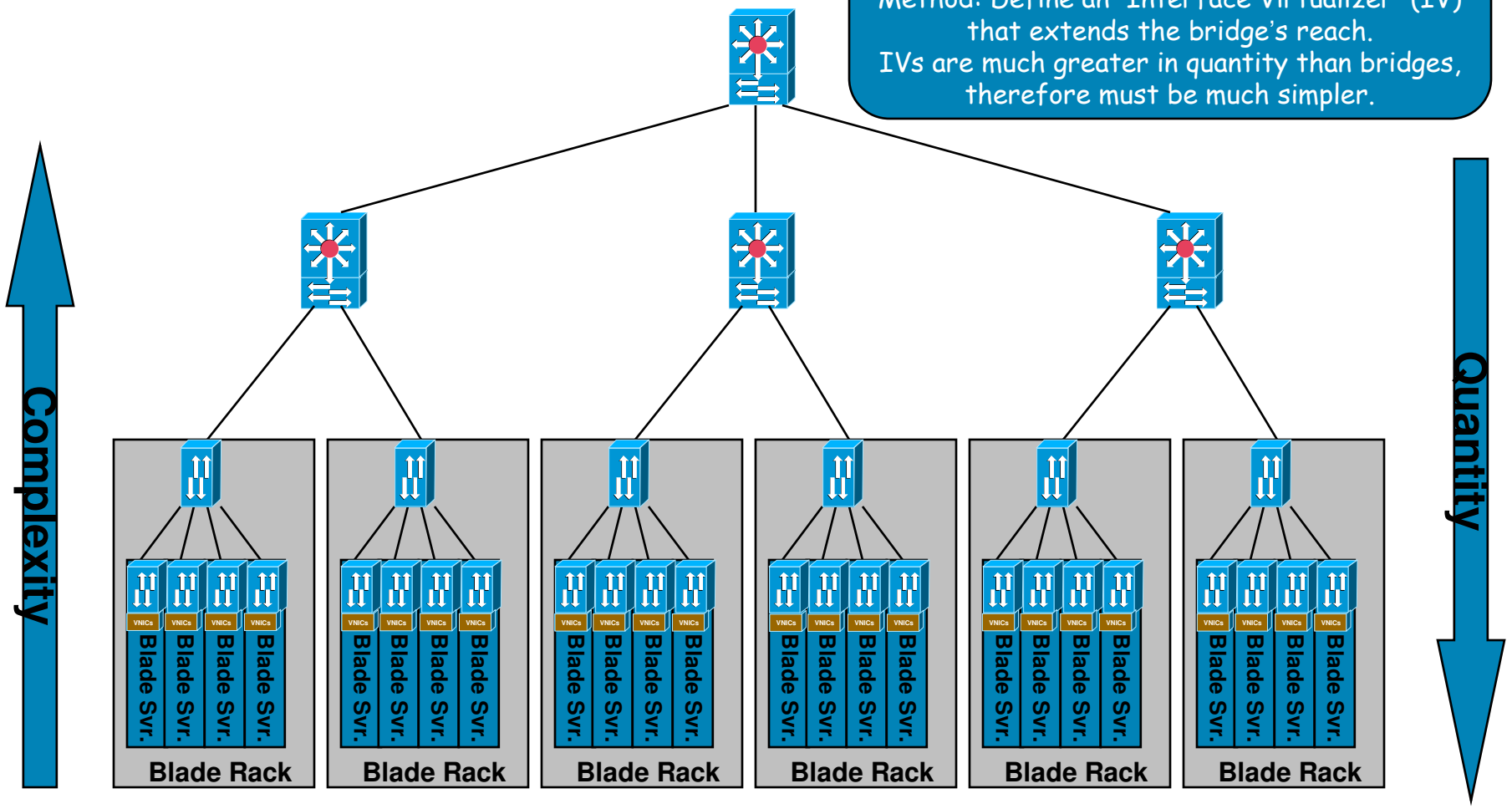
As virtualization and high density servers are deployed, we increase the number of complex bridges in excess of what use to be considered a large number of NICs

Even without virtualization, the same challenges exist. The sheer number of blade racks and 1U servers with their associated bridges is growing dramatically



Motivation

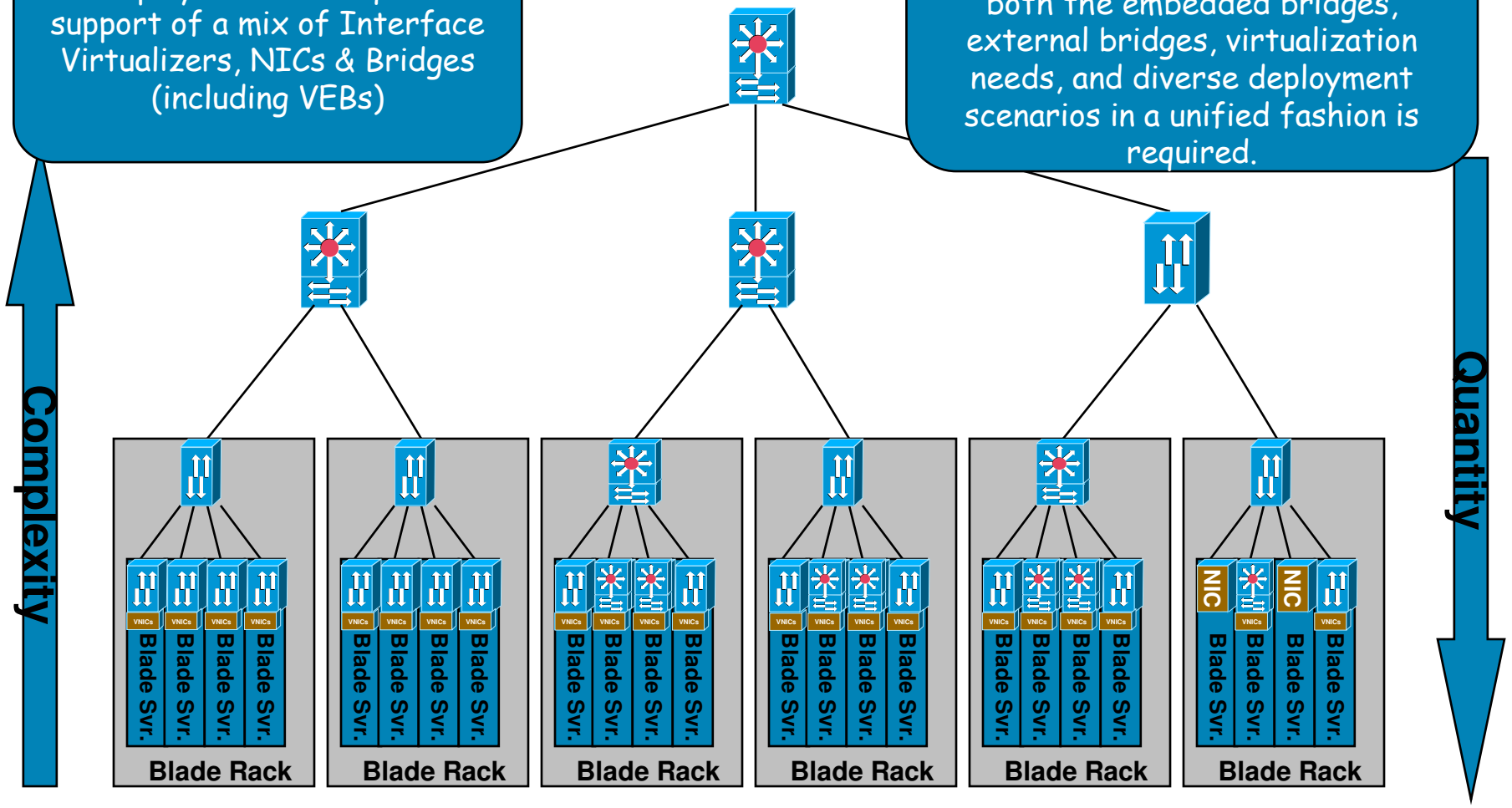
Goal: Extend the bridge into the blade racks and hypervisors, reducing the number of these complex devices.
Method: Define an "Interface Virtualizer" (IV) that extends the bridge's reach.
IVs are much greater in quantity than bridges, therefore must be much simpler.



Motivation

Deployment will require support of a mix of Interface Virtualizers, NICs & Bridges (including VEBs)

It is insufficient to address only the "embedded" portion of the problem. A solution that addresses both the embedded bridges, external bridges, virtualization needs, and diverse deployment scenarios in a unified fashion is required.



Problem Statement

- **The deployment of hundreds to thousands of bridge devices with diverse capabilities and performance as a result of high density server technology (including but not limited to server virtualization) creates the following challenges that are addressed by the proposed technology:**
  - **High network management complexity and administrative cost**
  - **High initial capital expenditures**
  - **Stressed scalability limits and responsiveness of network management applications due to:**
    - Volume of points of management
    - Volume of management messages required
- **Addressing just the embedded bridge in virtualized servers is insufficient to address the overall problem**
  - **Both embedded and external bridges contribute to the problems**
  - **One problem, one solution**

Some Initial Thoughts

- **There are essentially two proposals to be considered:**

  Tagged: VNTag

  Untagged: VEPA

- **Each provide certain capabilities**
- **Our goal is to analyze the each proposal to determine which provides the greatest benefit to cost ratio**

  This presentation will show the significant advantages that the tagged approach provides

  Detailed information on the VNTag proposal is provided; however, the main focus is to contrast tagging vs. non-tagging in general

    Details of tagging can be worked out later…

Requirements Summary

- **Must provide the same behavior to the station (i.e. NIC or VNIC) that is provided today by bridges**
  - **Fundamental to interoperability**
  - **Deviating from such behavior opens the door for unforeseen consequences**
  - **Extremely undesirable to require applications to be aware of whether they are directly connected to a bridge versus an Interface Virtualizer (or VEPA).**

Requirements Summary

- **Must be simple**
  - **Drive complexity towards the bridge and simplicity towards the NIC**
    - For example, ACL processing, CAM lookups, learning and aging functions, etc.
  - **Complexity should be limited to fewer devices**
    - Simplifies management
    - Lowers TCO
    - Simplifies upgrades
    - Etc.
  - **Avoid "two solutions to one problem issue"**
    - Consensus the VEB is a useful device
    - If we develop a device of similar complexity, cost, and management, there is little point
    - Simplicity provides the differentiation for use in the appropriate segments

Requirements Summary

- **Must operate in a variety of configurations**
  - **Downlinks must be able to connect to other Interface Virtualizers, bridges including VEBs, and NICs**
    - These devices may be virtual, instantiated together, or physically separate
  - **Focus specifically on the embedded function in an virtualized environment addresses only part of the problem**
    - Forcing us to address the external portions of the fabric with yet another solution
- **Must operate with existing applications and those in the foreseeable future, for example those:**
  - **That utilize various forms of ACLs**
  - **That depend on VLAN enforcement**
  - **That utilize MAC addresses other than those assigned by a hypervisor**

Requirements Summary

- **Must efficiently support embedded bridging**
  - **For example, VEB**
  - **Neither VNTag nor VEPA are appropriate for all applications**
- **Must efficiently support converged networking**
  - **These technologies expect certain functions commonly available in bridges today**
    - VLAN enforcement, locally assigned MAC addresses, basic ACL capabilities, static forwarding entries, etc.
  - **Must ensure these capabilities carry forward**

Requirements Summary

- **Must provide simple and efficient management capabilities**
  - **Reducing "points of management" is a good start**
    - However, if a "point of management" must initiate additional management messages, little has been gained
  - **Must provide predictable and consistent capabilities**
    - e.g. reduce fabric dependencies for VM migration and converged networking
  - **Reduce the number of devices that are "touched" by a management operation**
- **Must be cost effective**
  - **Otherwise there is no point…**
  - **The cost vs. benefit must be superior to other approaches**
- **Must minimize changes to bridge architecture**
  - **No need for invention for its own sake**
  - **Reuse proven technology and methods**

An Approach

An Approach

- **VNTag proposes to meet the previously stated requirements by providing a capability to combine distributed network components into a single logical 802.1Q compliant bridge**
- **These components consist of:**

  A centralized Controlling Bridge

  Distributed Interface Virtualizers (that may be cascaded)

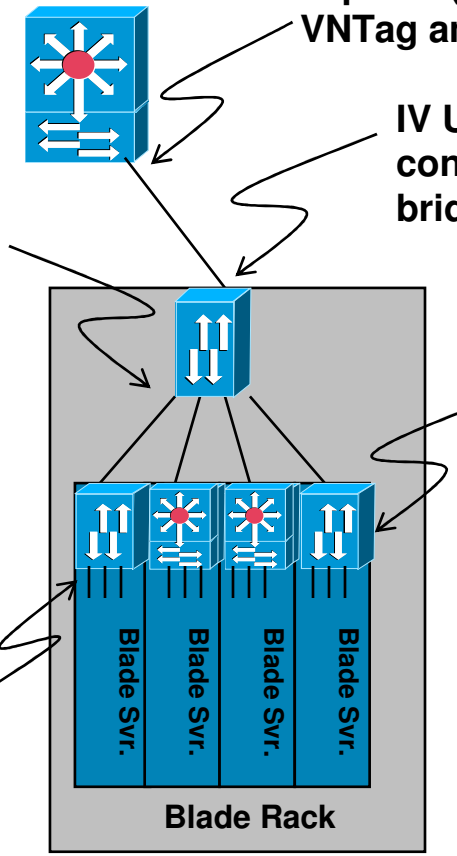
  A protocol enabling control of the Interface Virtualizers by the Controlling Bridge
- **The set of the Controlling Bridge and the Interface Virtualizers form a *single* 802.1Q compliant bridge**

An Approach - Anatomy of an VNTag Fabric

Bridges that connect to IV Uplink Ports must be IV capable (e.g. support the VNTag and the VIC Protocol).

IV Uplink Port: may connect to an IV capable bridge or an IV downlink

IV Downlink Port: may connect to an IV Uplink Port, a bridge or VEB, or a NIC (virtual or physical). Note that the bridge does not need to be IV capable in this case.



IVs may be cascaded. In this case, the Downlink Ports (virtual in this example) act as ports of the top level bridge.

Downlink ports are assigned a Virtual Interface Identifier (vif_id) that corresponds to a virtual interface on the bridge and is used to route frames down through IVs

Note: multiple Uplink Ports connected to different bridges or IVs are supported and are described later in this presentation.

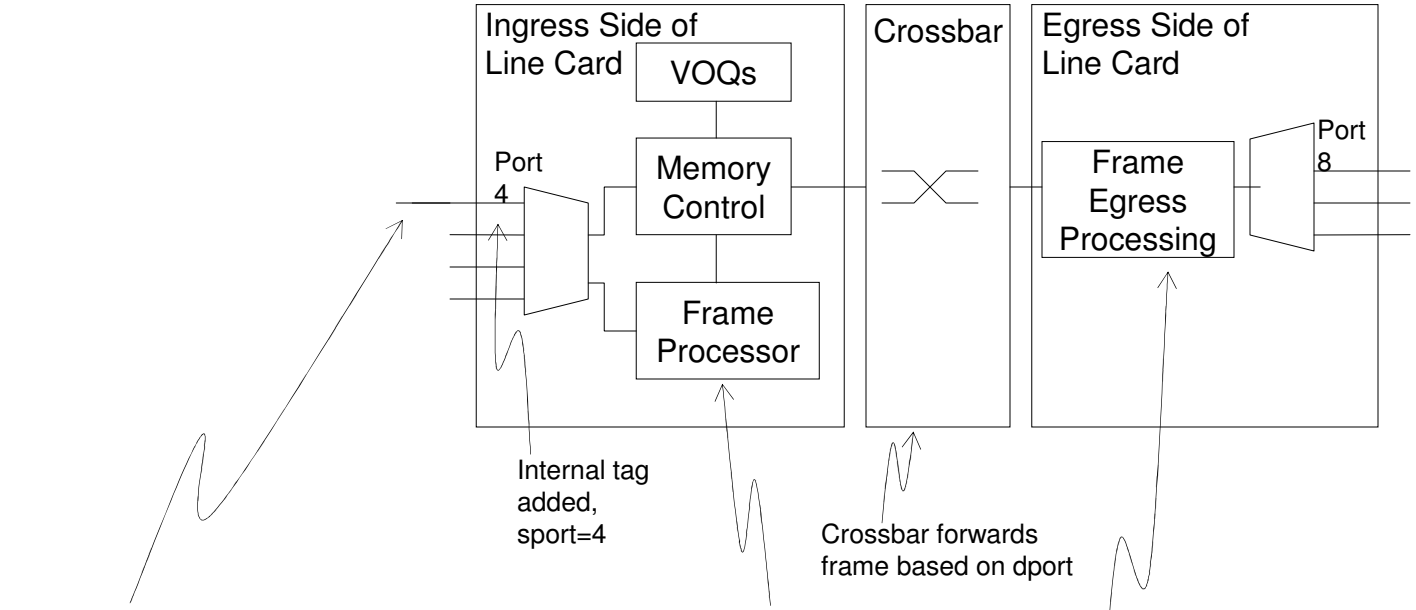
An Approach - Observations

- **To the greatest extent possible, all bridging functions are performed in the Controlling Bridge**
  - **Many bridging functions require knowledge of the ingress and/or egress port. A tag provides this information**
- **The ports on the south side of an IV are *physical* ports**
  - **You can see, touch, smell, and taste them**
  - **If you plug a network analyzer into one, it will see an 802.1Q compliant bridge**
    - The tags are limited between the IV and Controlling Bridge, so you would never see one at this point
  - **Inserting an IV is similar to inserting a line card**
    - New ports are instantiated in the Controlling Bridge just as if a line card was inserted
    - These ports are managed just as if they were part of a new line card
  - **There is nothing virtual about it!**
- **The ports of an embedded IV may be "virtual"**
  - **That is, they are conceptual and connect to a conceptual NIC (commonly referred to as a virtual NIC).**
  - **However, from the point of view of the Controlling Bridge and management of these ports, they are handled just like any other port**

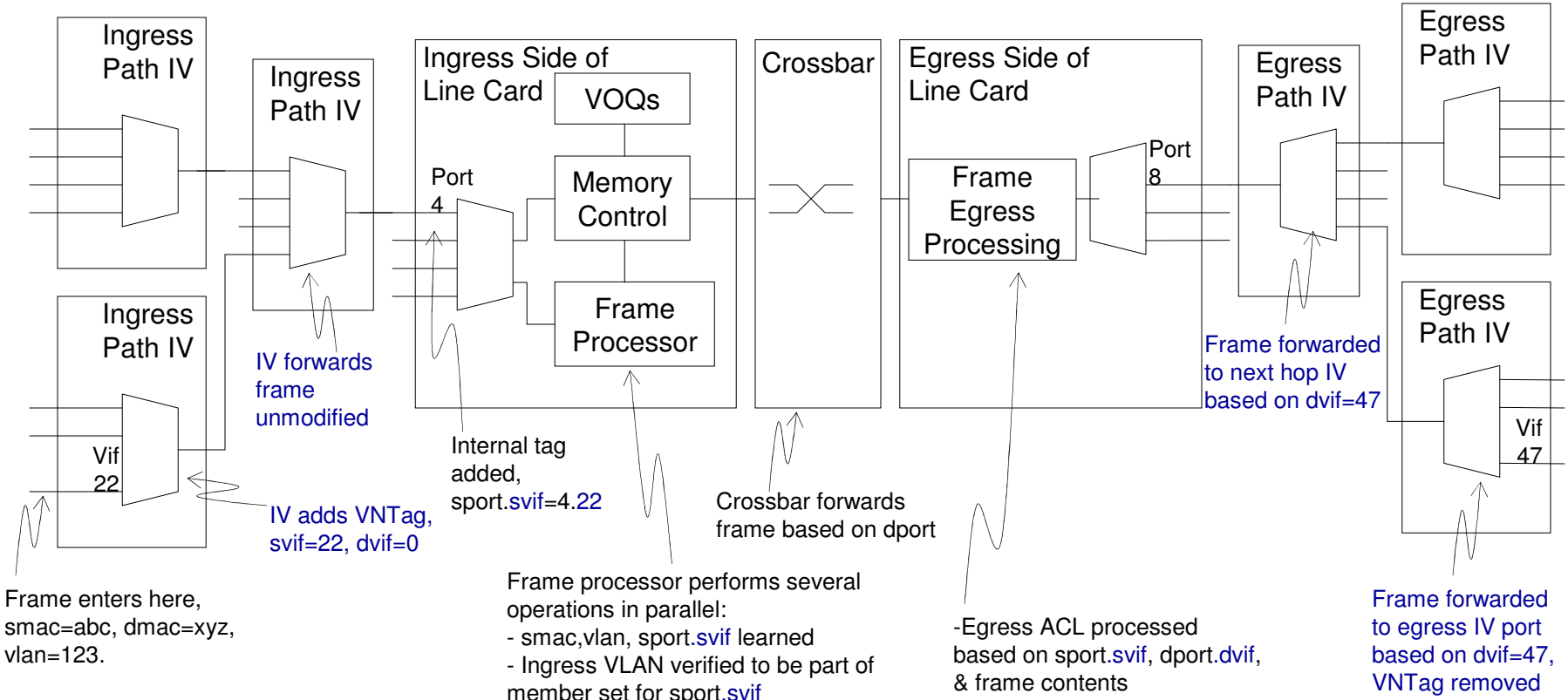
The Path to Tagging

Tagging is a natural extension of Bridge Functionality

The Path to Tagging



The Path to Tagging



The Path to Tagging - Observations

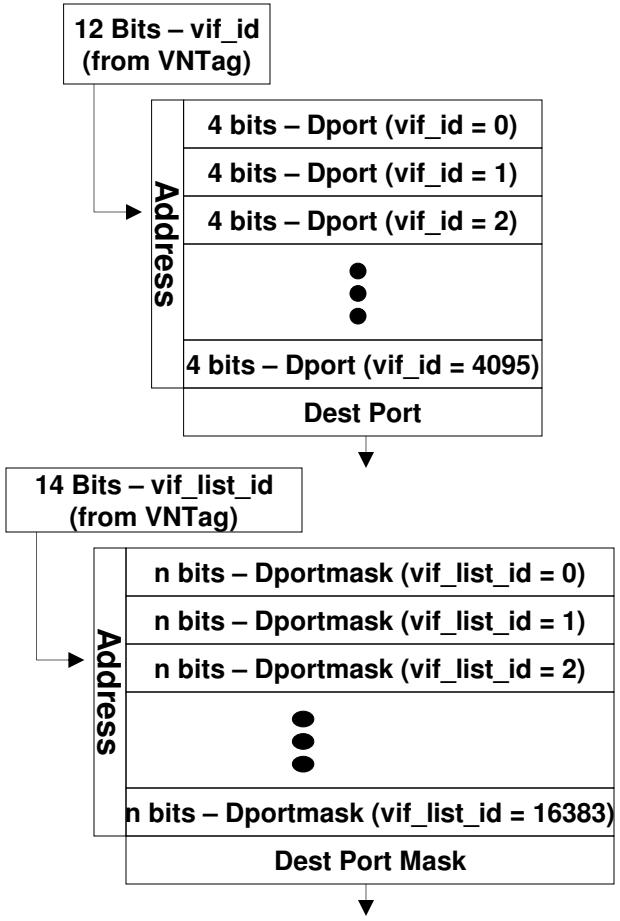
- When a frame enters a bridge, it is internally "tagged" with an indication of the ingress port
- The ingress port is used in several frame processing operations, ultimately resulting in determination of the egress port, which is added to the internal tag
- The rest of the forwarding through the bridge is performed based on the internal tag
- At egress, egress ACL processing is performed based on ingress port, egress port, and Frame Contents *(on a per egress port basis for multicast).* Frame processing adds or removes a QTag, and potentially other packet rewrite functions
- With VNTag, all of the fundamental bridge functionality remains identical
  - Which is a very good thing ☺
  - From the outside world, the combination of IVs and the controlling bridge is a single 802.1Q compliant bridge
- The IVs are extremely simple
  - On ingress, add a tag, then forward north
  - Southbound, forward based on vif_id as index into forwarding table
    - Remove VNTag at a last hop

The VNTag proposal

IV Downlinks & Virtual Interface Identifiers

- **Each downlink from an IV to a NIC, VNIC, bridge, or VEB is, in effect, a bridge interface**
  - **These are the instantiations of interfaces of the Controlling Bridge**
  - **Each downlink identified by a 12-bit Virtual Interface Identifier (vif_id)**
    - Assigned by the bridge to each IV downlink port at IV initialization
    - Scope of uniqueness is the Controlling Bridge Port

IV Forwarding Tables



- **VIF forwarding table (used for unicast)**
  - **One entry per VIF_ID**
    - May support up to 4096 unique VIF_IDs
    - Indexed by Dvif_id (part of the VNTag)
    - Each entry points to the downlink to be used
- **Vif list table (used for multicast and flooding)**
  - **One entry per vif_list_id**
    - May support up to 16k unique lists
    - Indexed by vif_list_id
    - Each entry contains a bit mask indicating which downlinks are to be used
      - Width of entry depends on number of downlink ports
- **Note: Table size not a function of VLANs / MAC addresses in use**
  - Each interface utilizes a single entry regardless of the number of VLANs and / or MAC addresses in use on that interface

Interface Virtualizer Basic Functions

- **From NIC to Controlling Bridge**
  - **Add VNTag if none present (indicating source vif_id)**
    - VNTag added *only* at ingress
      - VNTags are not “stacked” as the frame passes through successive IVs
  - **Forward frame up the IV hierarchy to the Controlling Bridge**
- **From Controlling Bridge to NIC**
  - **Froward frame down hierarchy to the NIC**
    - Destination port determined by using Dvif_id as index into the forwarding table
  - **Replicate multicast frames**
    - Filter the frame at the ingress port if it was sourced at the IV
      - (i.e. if the port’s assigned vif_id matches the source vif_id in the VNTag)
  - **Remove the VNTag if the final downlink has been reached**

Bridge use of VN_Tag

- **On ingress**

  Learn vif_id along with MAC address, VID, and port number as part of normal bridge learning function

- **Forwarding**

  Utilize source vif_id along with ingress port number as frame source for all normal bridge functions (ACLs, VLAN member set enforcement, etc.)

- **On egress:**

  Populate the VNTag with the source and destination vif_ids

Support of Bridge and other PDUs

- **The set of a Controlling Bridge and its Interface Virtualizers form an 802.1Q compliant bridge**

  Implies that the Controlling Bridge must have the ability to send arbitrarily addressed protocol frames to specific IV egress ports (e.g. BPDUs), and to identify from which port these frames were received (independent of source MAC address)

  Solution:

  For transmission, address the protocol frame as appropriate, and direct it to the desired IV egress port with an appropriate VNTag

  On reception, the VNTag provides the identity of the port from which the frame was received

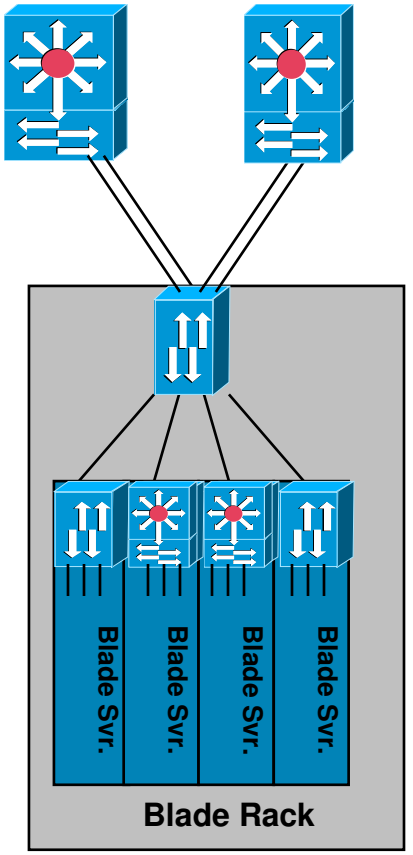
- **Note: this is the same function that is performed internal in bridges today**

  Every frame received by the bridge's control processor is somehow marked with an ingress port number indication

  Every frame transmitted by the bridge's control processor is somehow marked with an egress port number inidcation

Support of Multiple Uplink Ports

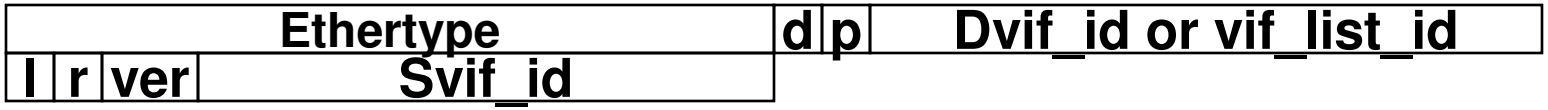
- **Required for:**
  - **Redundancy**
  - **Support of multiple fabric connectivity**
- **Achieved by:**
  - **Instantiating a VIF forwarding table and VIF list table for each uplink port**
    - Addresses “Southbound” frames
  - **Each downlink port is associated with a single uplink port**
    - All frames received on that downlink port are forwarded to the associated uplink port
    - Addresses “Northbound” frames



Additional Interface Virtualizer Functions

- **QTags are always present on uplinks**
  - **On IV ingress, if QTag not present**
    - Add QTag with VID=PVID and Pri=0
    - If Priority Tagged, insert PVID into QTag
  - **On IV egress, if egress port is in VIDs untagged set**
    - Remove QTag
    - Note: in actual implementation, it is allowable and common to support a single untagged VLAN per egress port. In this case, the QTag is removed if the VID matches the programmed value for that port (often this is simply the port's PVID value).
- **DCB Functions**
  - **Priority-based Flow control**
  - **Enhanced Transmission Selection**
  - **Congestion Notification**
  - **DCBX: Provided by Controlling Bridge**
- **Advanced Functions**
  - **None: Interface Virtualizers are simple**

VNTag Proposal



Ethertype:	**(16 bits), TBD, identifies the VNTag**
d:	**Direction (1 bit), 0 indicates that the frame is traveling from the IV to the bridge. 1 indicates the frame is traveling from the bridge to the IV**
p:	**Pointer (1 bit): 1 indicates that a vif_list_id is included in the tag. 0 indicates that a Dvif_id is included in the frame**
vif_list_id:	**Vif List Pointer (14 bits), points to a list of downlink ports to which this frame is to be forwarded (replicated)**
Dvif_id:	**Destination vif_id (2 reserved bits, 12 used bits) of the port to which this frame is to be forwarded. Two most significant bits are reserved.**

Note: the Dvif_id / vif_list_id field is reserved if d is 0.

l:	**Looped (1 bit): 1 indicates that this is a multicast frame that was forwarded out the bridge port on which it was received. In this case, the IV must check the Svif_id and filter the frame from the corresponding port**
r:	**(1bit) reserved**
ver:	**(2 bits) Version of this tag, set to 0**
Svif_id	**The vif_id (12 bits) of the downlink port that received this frame from the VNIC (i.e. the port that added the VNTag). This field is reserved if d=1 and l=0.**

Virtual Interface Control (VIC) Protocol

- **Controlling Bridge configures all of the forwarding tables for each downstream (i.e. cascaded) IV**

  Occurs at IV initialization

  No additional programming required as the result of MAC learning / aging, or MAC migration as the result of VM migration

- **VIC Protocol provides this functionality**

  Low overhead reliable L2 transport

  All messages are command / response

    All commands are idempotent enabling repeatability if command or response is lost

  Independent instance of VIC is executed for each Uplink Port (or Uplink Port Aggregation)

VIC Controller and associated addressing

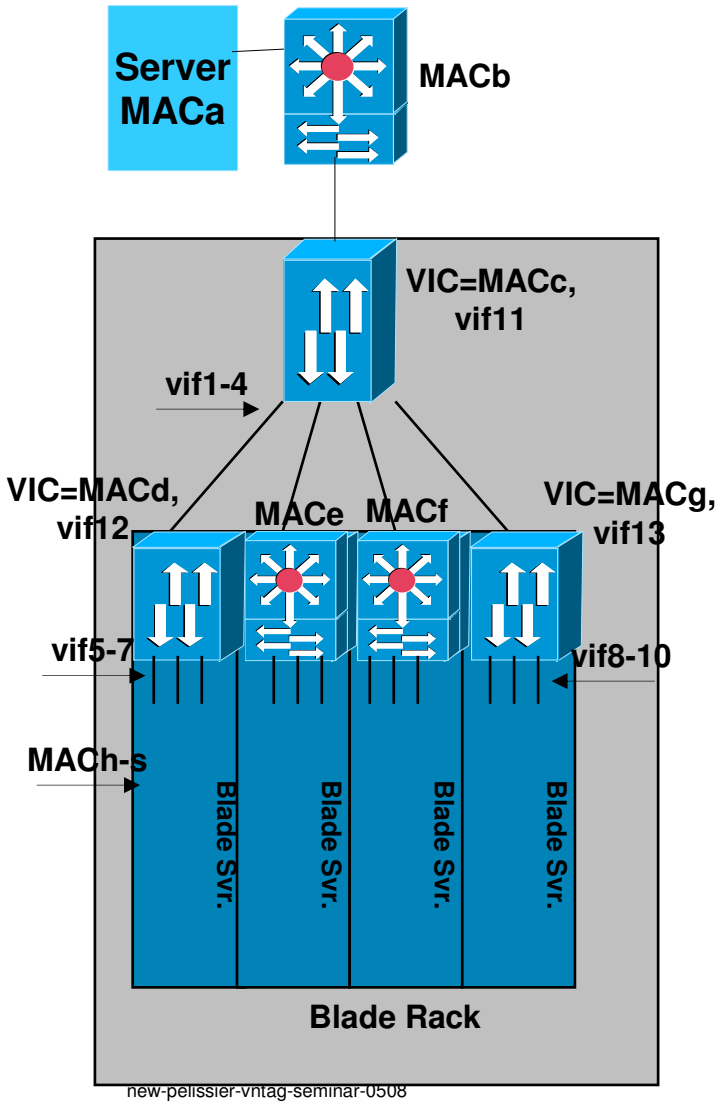
- VIC Controller is the entity within an Interface Virtualizer that executes the VIC protocol
- Addressed using its unique MAC address and vif_id
  - VNTag routes frame through cascade of IVs to proper VIC Controller
  - Vif_id assigned to VIC Controller using a “bootstrap” protocol
    - DCBX, for example

Basic VIC Operations

- Open: Establishes link between bridge and an NIV
- Create: Sent by an IV requesting bridge to create a new interface
- Delete: Sent by an IV requesting bridge to delete an interface
- Enable: Sent by an IV requesting bridge to enable an interface
- Disable: Sent by an IV requesting bridge to disable an interface
- Set: Sent by bridge indicating that a VIF has been enabled and the state (e.g. vif_id) that is to be used by the corresponding downlink port in the IV. May also be used by the bridge to inform the IV that an interface has gone down.
  - In a cascaded arrangement, a set is sent to each IV in the cascade to program the forwarding tables
- Get: Sent by bridge or IV to obtain the interface state of a peer
- List set & list get: programs / retrieves the vif list tables in IVs

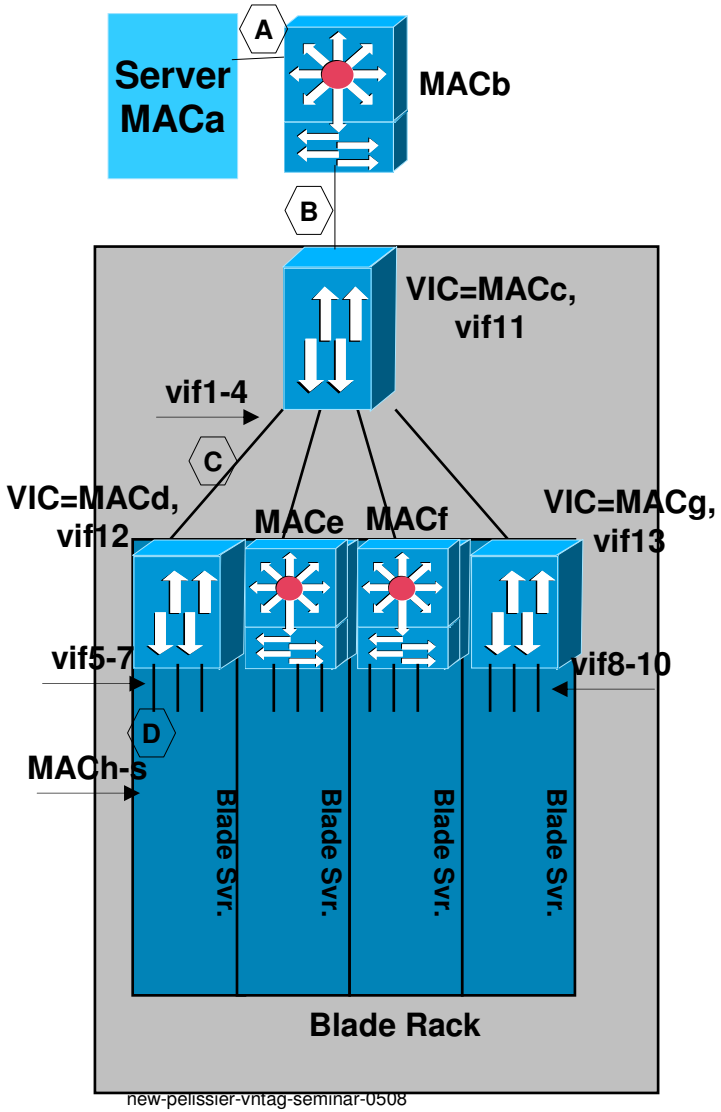
VNTag Addressing Examples

Addressing Examples Overview



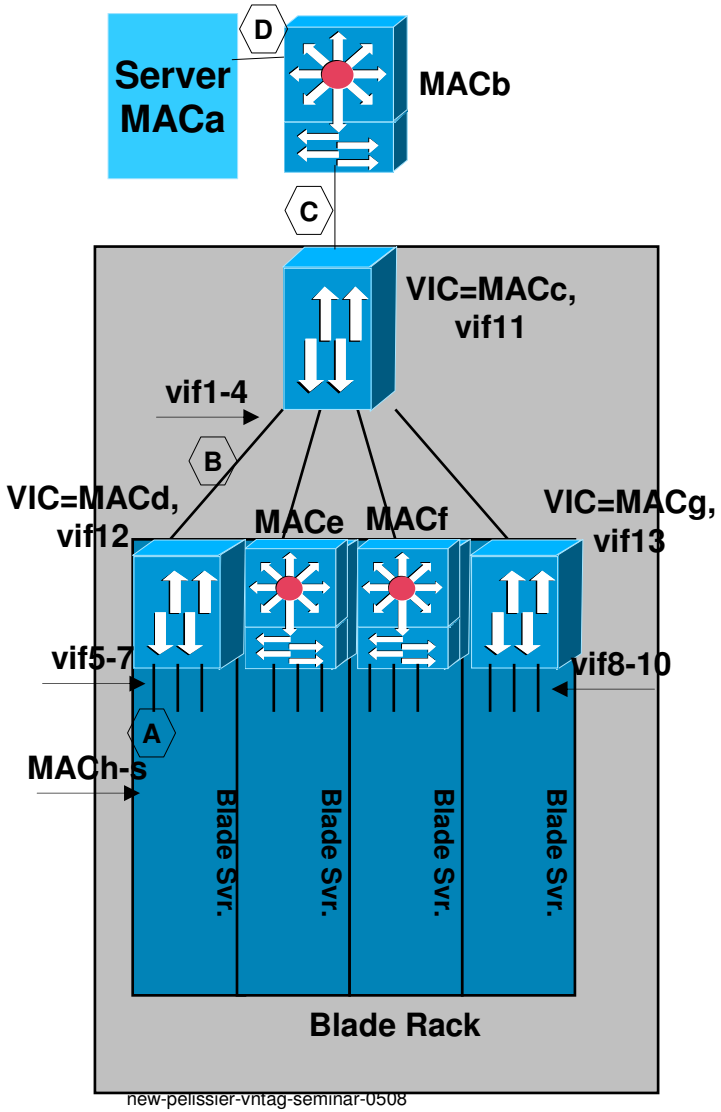
- Top Server has MAC address MACa
- Top bridge has MAC address MACb
- VIC Controller associated with the uplink in the top IV has MAC address MACc and interface id vif11
- The downlinks on the top IV have virtual interface ids vif1 through vif4
- The VIC controllers associated with the uplinks in the two blade servers have MAC addresses MACd and MACg and interface ids vif12 and vif13.
- The two bridges in the blade servers have MAC addresses MACe and MACf.
- The downlinks on the IVs in the blade servers have virtual interface ids vif5 through vif10
- Each blade server has three VMs. The MAC addresses of the VMs are MACh through MACs from left to right

Addressing Examples



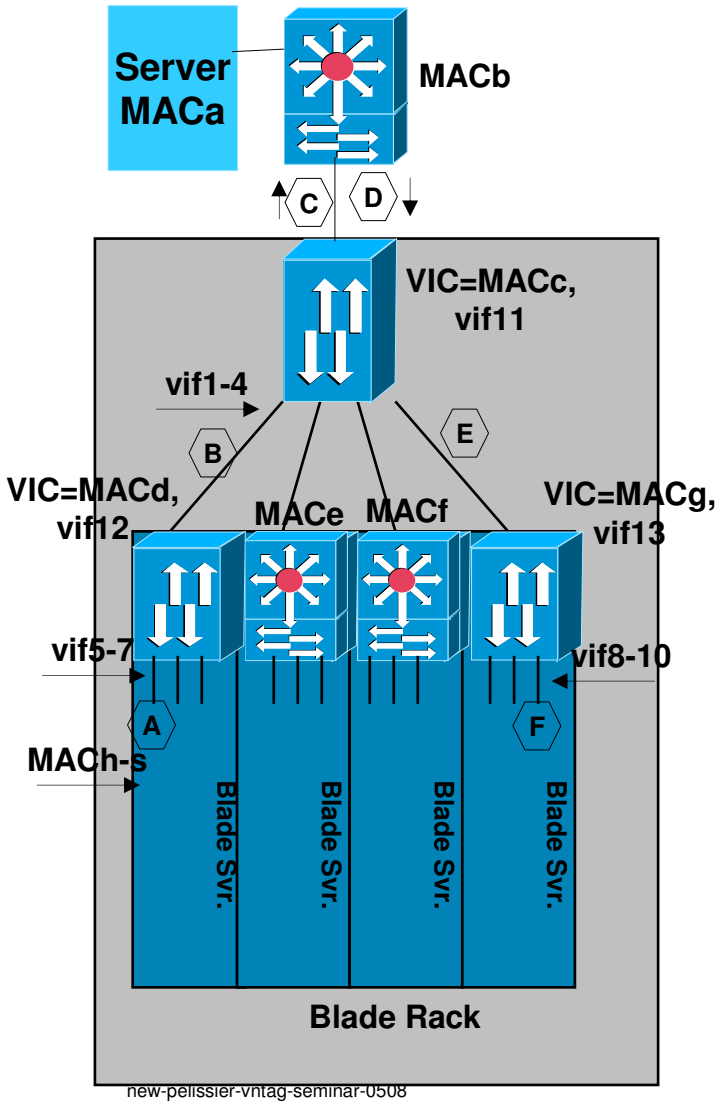
Unicast Frame from Server at MACa to VM at MACh					
Location	DA	SA	VNTag Present?	Dvif	SVif
A	MACh	MACa	No		
B	MACh	MACa	Yes	vif5	none
C	MACh	MACa	Yes	vif5	none
D	MACh	MACa	No		

Addressing Examples



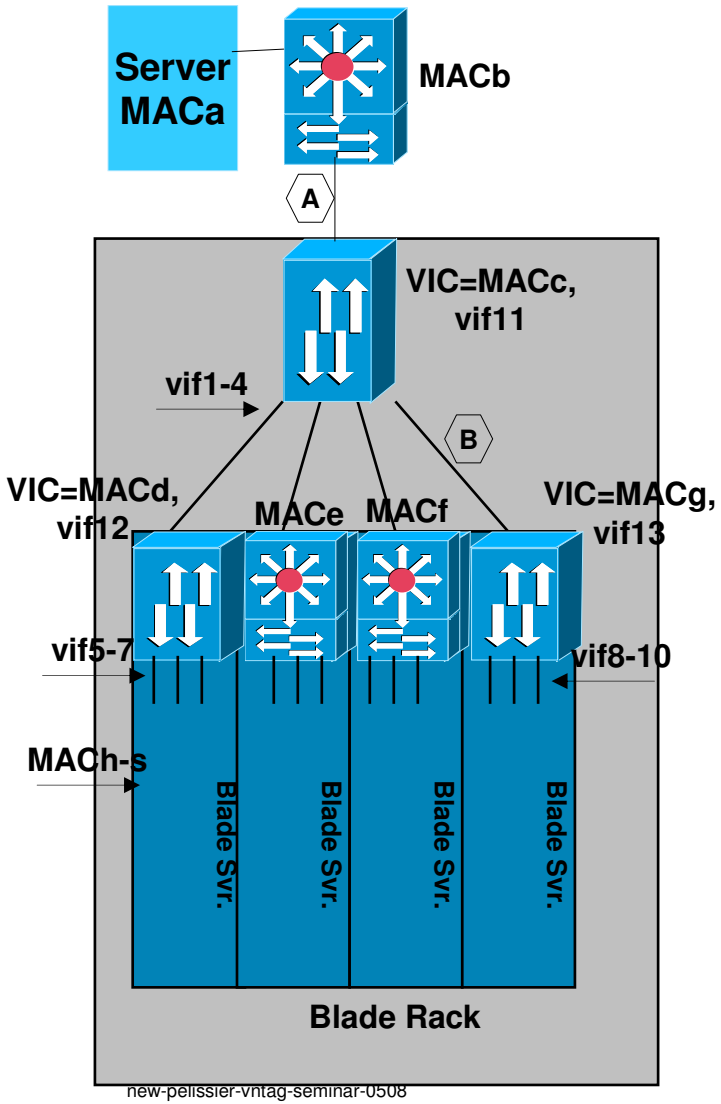
Unicast Frame from VM at MACh to Server at MACa					
Location	DA	SA	VNTag Present?	Dvif	SVif
A	MACa	MACh	No		
B	MACa	MACh	Yes	none	vif5
C	MACa	MACh	Yes	none	vif5
D	MACa	MACh	No		

Addressing Examples



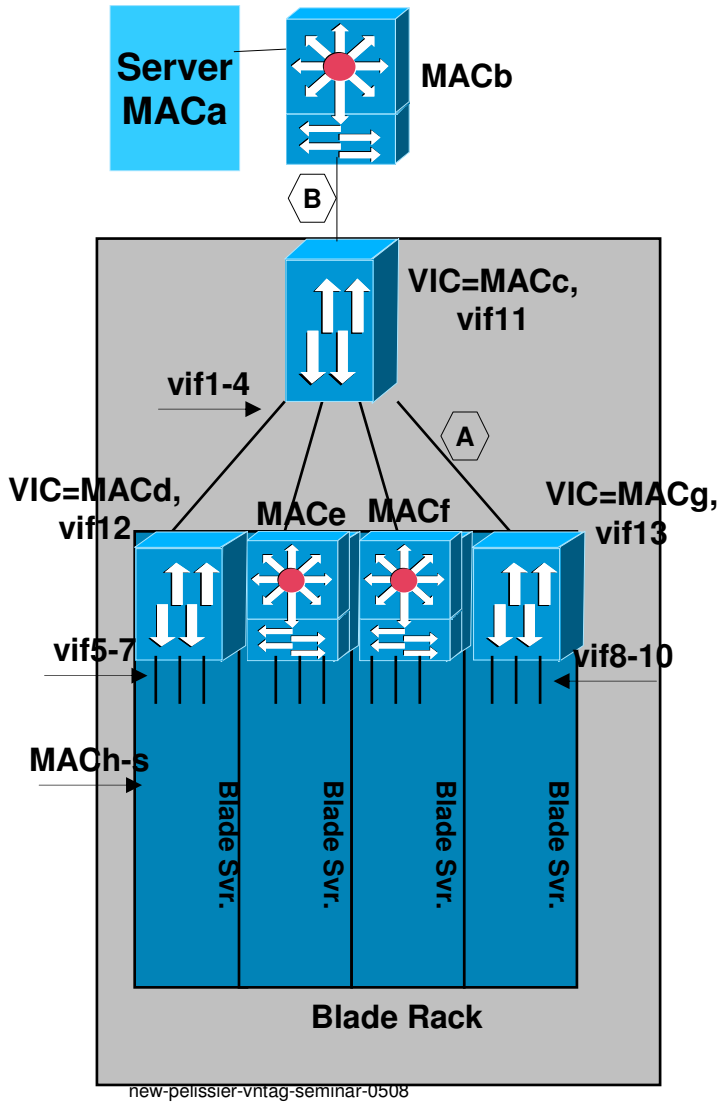
Unicast Frame from VM at MACh to VM at MACs					
Location	DA	SA	VNTag Present?	Dvif	SVif
A	MACs	MACh	No		
B	MACs	MACh	Yes	none	vif5
C	MACs	MACh	Yes	none	vif5
D	MACs	MACh	Yes	vif10	vif5
E	MACs	MACh	Yes	vif10	vif5
F	MACs	MACh	No		

Addressing Examples



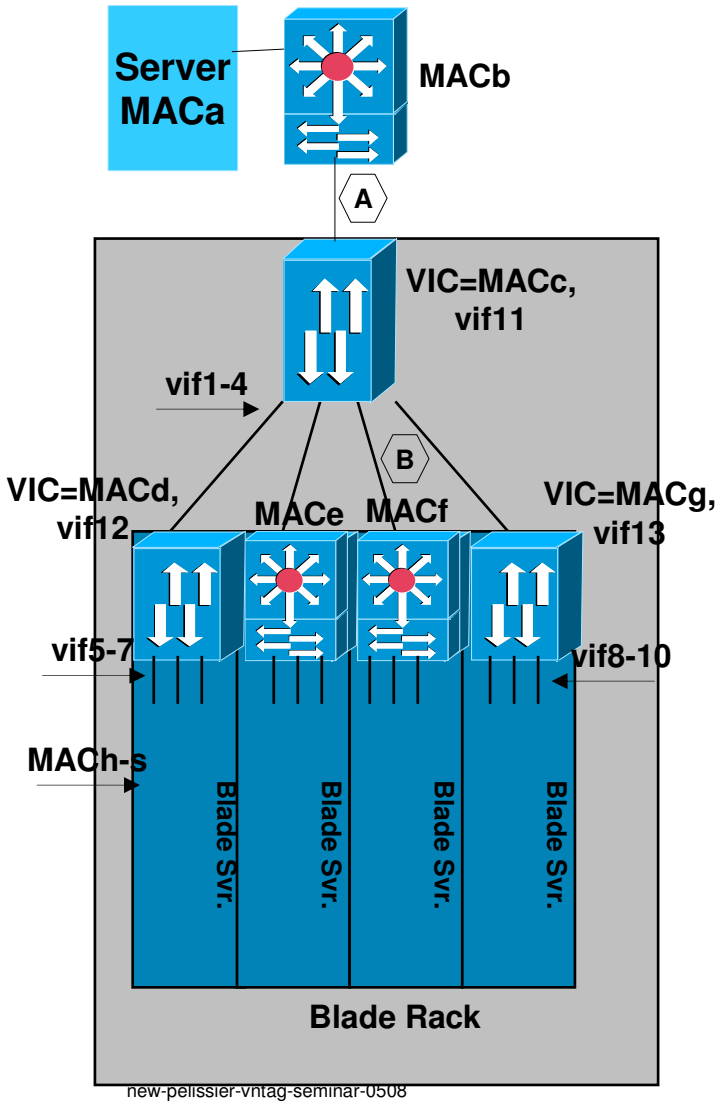
Unicast VIC control frame from bridge at MACb to VIC Controller at MACg					
Location	DA	SA	VNTag Present?	Dvif	SVif
A	MACg	MACb	Yes	vif13	none
B	MACg	MACb	Yes	vif13	none

Addressing Examples



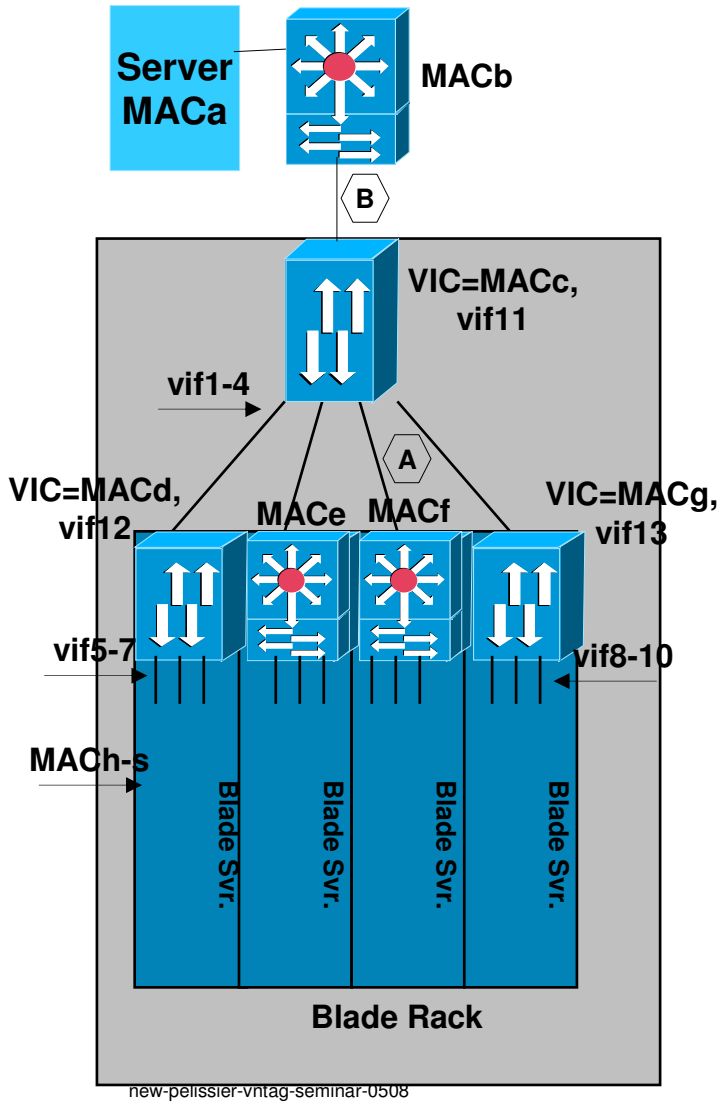
Unicast VIC control frame from VIC Controller at MACg to bridge at MACb					
Location	DA	SA	VNTag Present?	Dvif	SVif
A	MACb	MACg	Yes	none	vif13
B	MACg	MACb	Yes	none	vif13

Addressing Examples



BPDU from bridge at MACb to bridge at MACf					
Location	DA	SA	VNTag Present?	Dvif	SVif
A	01-80-c2-00-00-00	MACb	Yes	vif3	none
B	01-80-c2-00-00-00	MACb	No		

Addressing Examples



BPDU from bridge at MACf to bridge at MACb					
Location	DA	SA	VNTag Present?	Dvif	SVif
A	01-80-c2-00-00-00	MACf	No		none
B	01-80-c2-00-00-00	MACf	Yes	None	vif3

Case Studies

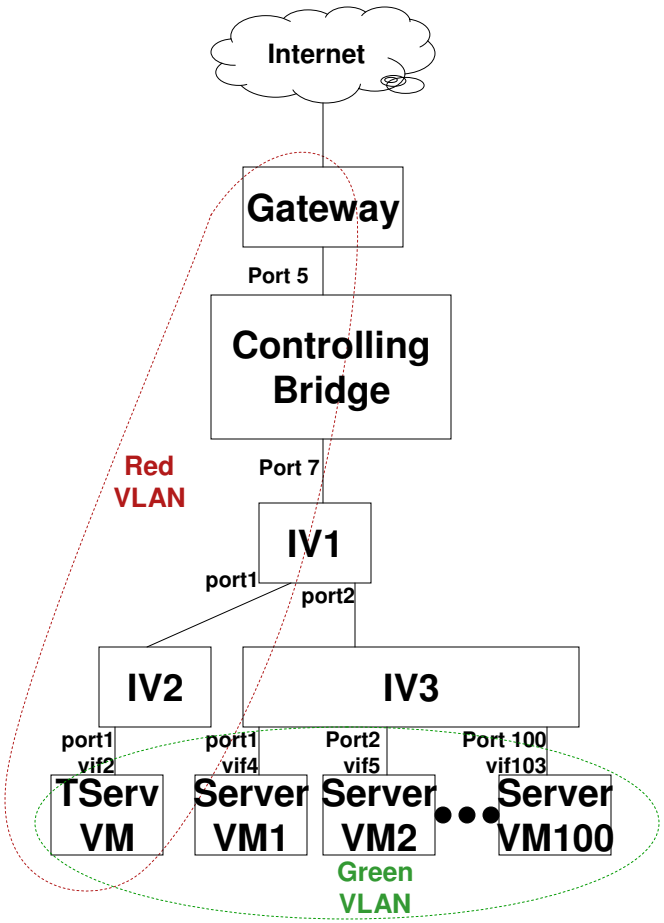
Address Learning and Forwarding:

Transparent Services Example

Address Learning & Forwarding Case Study - Background

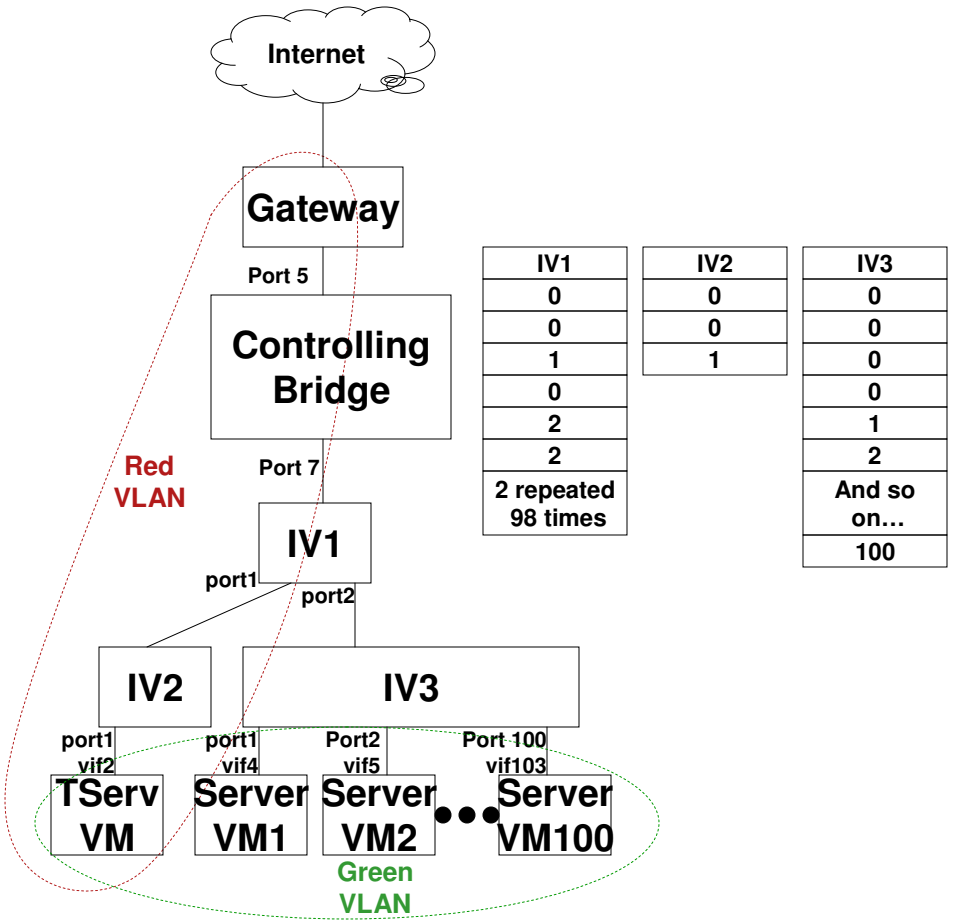
- **Transparent Services refers to various critical data center services including:**
  - **Firewalls**
  - **Load balancers**
  - **Intrusion detection and prevention**
  - **Policy compliance monitoring**
  - **Etc.**
- **These services are "transparent" in that they do not generate their own traffic**
  - **They are inserted in the network and traffic transparently flows through them**
    - Thus these services are able to monitor all of the traffic and perform their respective functions

Address Learning & Forwarding Case Study – Transparent Service Insertion



- The Transparent Service (TServ) utilizes two VLANs
  - The Red VLAN contains the untrusted data between the gateway and TServ
  - The Green VLAN contains the trusted data between the TServ and all of its clients
- In normal operation, TServ does not modify the frames passing through it
  - On the Green VLAN, the TServ appears to have the MAC address of the Gateway
  - On the Red VLAN, the TServ appears to be a bridge device emitting frames with every MAC address from the Green VLAN

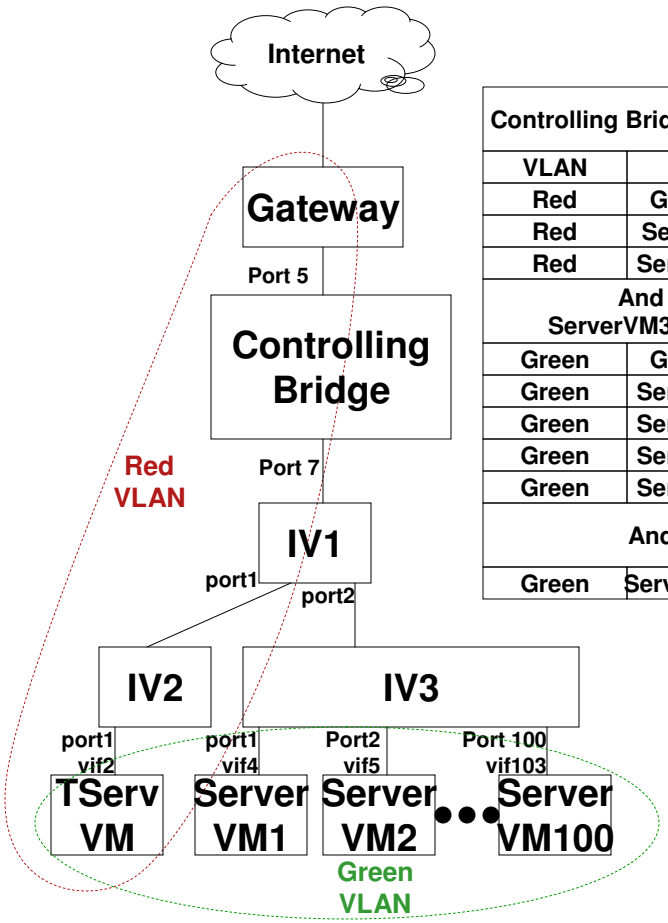
Address Learning & Forwarding Case Study – Transparent Service Insertion



- **Upon Initialization, the IV forwarding tables are programmed as indicated.**
- **These tables remain static**

  They are not updated as new MAC addresses / VLANs are learned

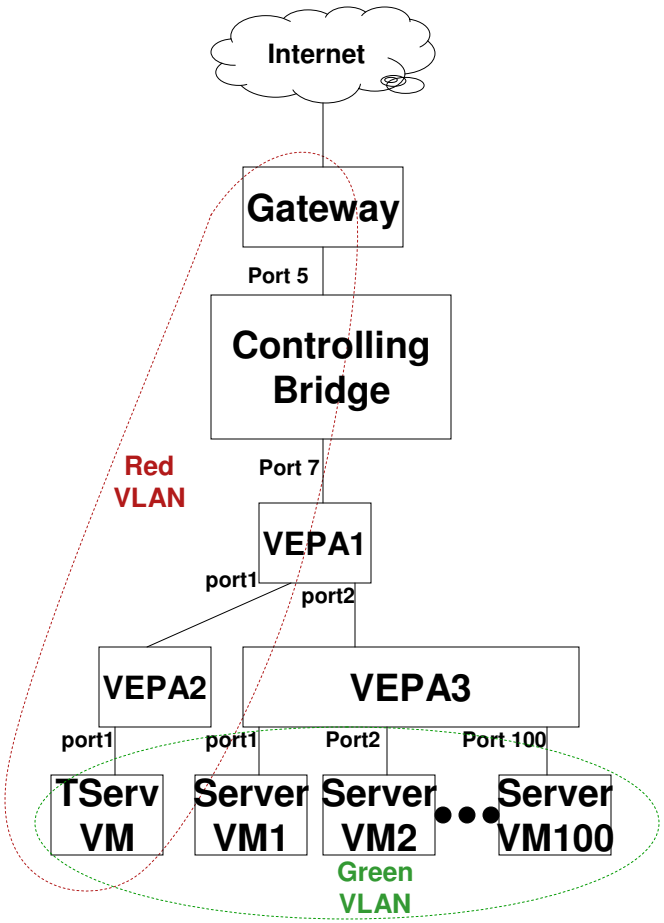
Address Learning & Forwarding Case Study – Transparent Service Insertion



Controlling Bridge Forwarding Table		
VLAN	MAC	Port
Red	Gateway	5
Red	SeverVM1	7.2
Red	ServerVM2	7.2
And so on for ServerVM3 – ServerVM100		
Green	Gateway	7.2
Green	ServerVM1	7.4
Green	ServerVM2	7.5
Green	ServerVM3	7.6
Green	ServerVM4	7.7
And so on…		
Green	ServerVM100	7.103

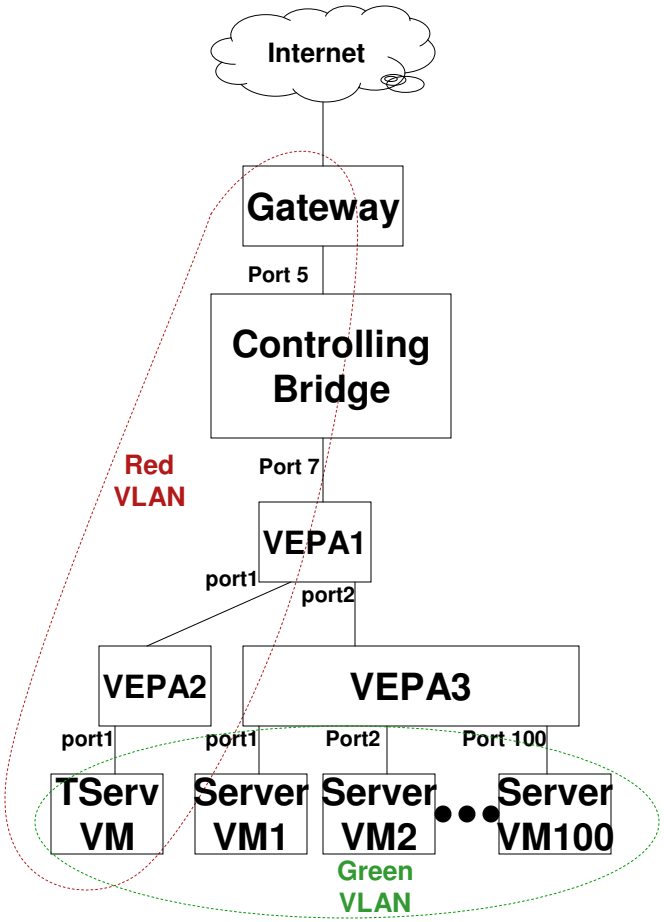
- **During operation, the Controlling Bridge learns MAC addresses in the normal manner**
  - **In addition, it learns the corresponding vif_ids**
- **The Controlling Bridge forwarding table will converge as indicated**
  - **Note that the TServ has no MAC address for itself**
    - (actually it does for management, etc., but that is not relevant to this discussion)

Address Learning & Forwarding Case Study – The Untagged Dilemma



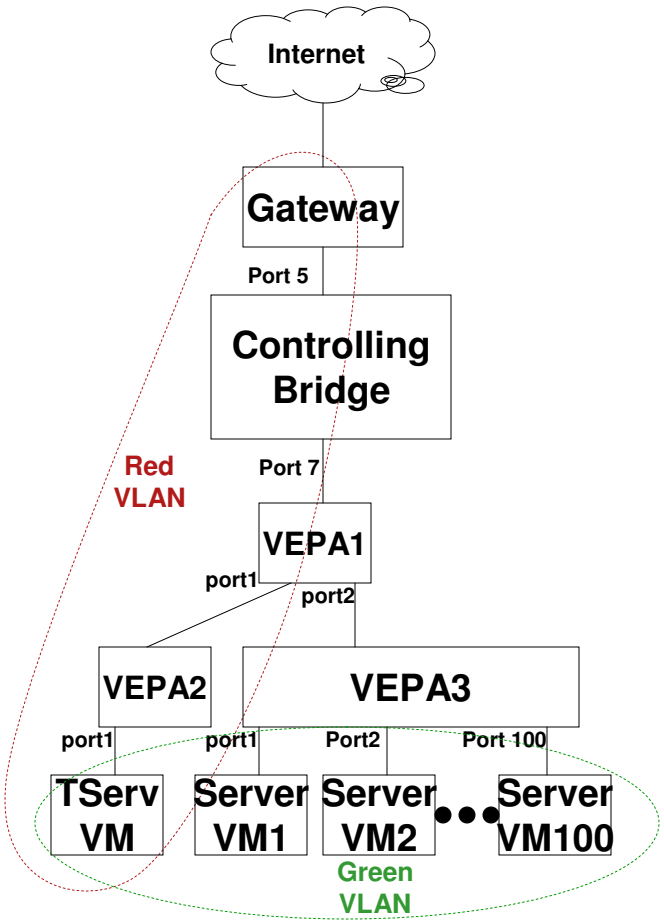
- VEPAs do not learn MAC addresses
  - They are told the MAC addresses by the attached hypervisors
- In this case, the addresses seen coming out of the TServ were not assigned by the hypervisor
  - In fact, they are addresses assigned by another hypervisor, but on a different VLAN
- Therefore, an API would need to be created that would allow a VM to tell the hypervisor what addresses it is using
  - The VNICs are operating in promiscuous mode, thus these addresses would not normally be "configured" by the VM into the VNICs

Address Learning & Forwarding Case Study – The Untagged Dilemma



- **Even with the API this is incredibly inefficient:**
  - **For every frame received from the green VLAN:**
  - **-Has hypervisor been informed; if not:**
  - **-Delay transmission of frame**
  - **-Inform hypervisor**
  - **-Hypervisor informs both VEPAs**
  - **-Hypervisor informs VM that VEPAs ready**
  - **-TServ then clear to transmit frame**
    - -Meanwhile, frames back up and are discarded
- **Implies TServ must be "VEPA aware"**
  - **Must be prepared to inform hypervisor of MAC addresses**

Address Learning & Forwarding Case Study – The Untagged Dilemma



- **VM migration appears to leave residual entries in the VEPA**
  - **Assume VM2 moves to VEPA4**
  - **TServ has no visibility into this move**
    - Therefore, it will not be able to tell VEPA2 to invalidate the MAC address on the Red VLAN
  - **Analogous to a memory leak**
- **Less scalable**
  - **Every MAC / VLAN combination that appears on any VEPA port generates a forwarding table entry**
- **From a practical perspective, a VEPA would need to learn / age to support this class of applications**
- **Alternatively, the VEPA could tell the Controlling Bridge on which port the frame was received, and allow the Controlling Bridge to perform learning in the traditional fashion**

Case Studies

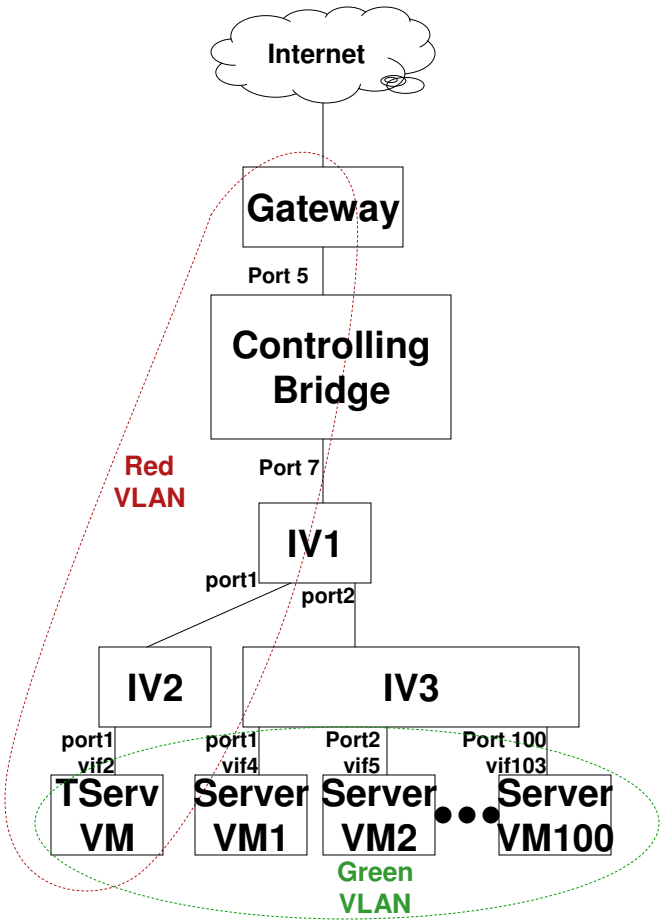
Ingress VLAN enforcement:

Transparent Services Example

Ingress VLAN Enforcement Case Study - Background

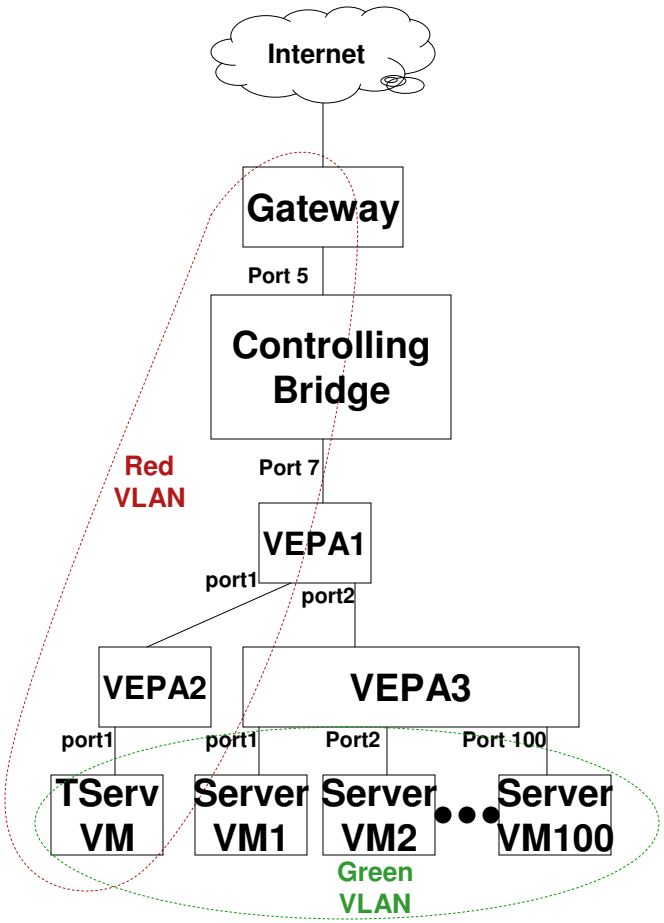
- 802.1Q provides an optional ability, on a per port basis, to restrict frame admittance to a given set of VLANs
  - Each VLAN has a “member set”, i.e. the set of ports that belong to the given VLAN
  - If the parameter “Enable Ingress Filtering” is set, then the ingress port is to perform the filtering.

Ingress VLAN Enforcement Case Study – Transparent Service Insertion



- The robustness of this deployment may be enhanced through the use of VLAN enforcement:
  - Port 5 should admit frames only on the red VLAN
  - Port 7.2 should admit frames on either the red or green VLAN
  - Port 7.4 through 7.103 should admit frames on only the green VLAN
- Without this enforcement, a Server could emit / receive frames on the Red VLAN bypassing the protections provided by the services
- In the controlling bridge, this is accomplished by the following member sets:
  - Red VLAN – ports 5 and 7.2
  - Green VLAN – ports 7.2, 7.4-7.103
- Since the frames arrived VNtagged, the Controlling Bridge can enforce these sets

Ingress VLAN Enforcement Case Study – The Untagged Dilemma



- **In this case, the controlling bridge does not know the ingress port; it cannot apply the appropriate member set**
  - **Note that the MAC address is an insufficient proxy for this since the same MAC address appears on multiple VEPA ports**
- **Thus, the ingress filtering must be performed in the VEPA**
  - **This is a simple process, but requires large amounts of memory**
    - e.g. a 4k bit mask for each virtual port
    - Note: this cannot be done as part of the VLAN/MAC lookup function for several reasons including the fact that no such lookup is performed northbound.
- **Alternatively, the VEPA could tell the controlling bridge on which port a frame was received, and then the controlling bridge can take care of the filtering**

Case Studies

Access Control Lists: FCoE Example

ACL Case Study - Background

- **Implementation of Access Control Lists are not standardized and capabilities vary widely across implementations**
- **However, ACLs are widely deployed to enhance the robustness on networks**
- **In general, ACLs:**
  - **Consist of an ordered set of rules that determine if a frame is to be forwarded (i.e. "permit") or discarded (i.e. "deny")**
  - **Each rule defined by matching bits in the received frame to a specified pattern**
  - **If multiple rules match, the first in the ordered list applies**
    - Not just pattern matching; implies TCAM or equivalent
  - **A default permit or deny may be specified**
  - **May be implemented at ingress, egress, or both**
  - **Specified on a per port basis**

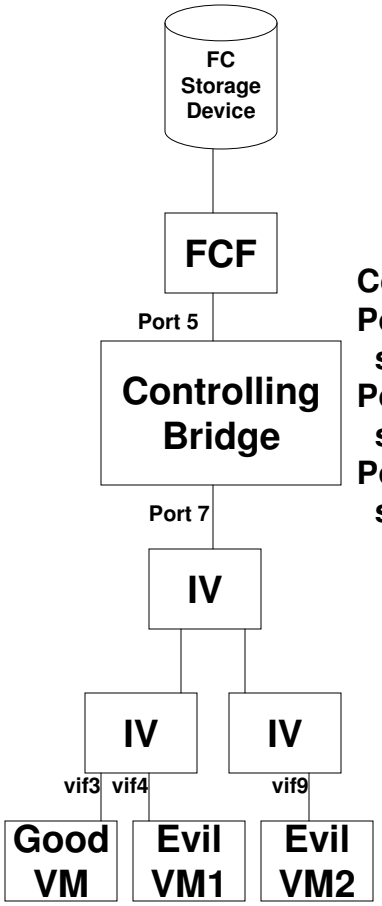
ACL Case Study - FCoE

- **FCoE utilizes ACLs to achieve robustness equivalent to native Fibre Channel**
- **In FC, it is not possible to "impersonate" another station**
  - **All FC devices utilize point to point links between the device and the FC switch**
  - **The FC switch assigns the device a Fibre Channel ID (FCID) at log in**
  - **The switch enforces use of that FCID in the source address of all frames received on that port**
- **With FCoE, intervening bridges allow multiple devices to appear on a given FCF port (the FCoE equivalent of a Fibre Channel Switch)**
  - **The FCF cannot enforce proper use of the source MAC address**
- **Impersonation enables attacks that can result in undetected data corruption and undetected data intercept**
  - **These attacks are easily thwarted using the most basic of ACL implementations**
    - Although processing of an ordered list is required

ACL Case Study – FCoE ACL

- **To protect against this attack, an ACL is installed at each edge bridge port that:**
  - **Permits frames whose source address matches that assigned by the FCF**
  - **Denys frames whose source address matches any other MAC address assigned by an FCF anywhere in the network**
    - FCF assigned MAC addresses are identified by the fact that the 24 most significant bits match an FCoE configured constant FC-MAP
- **The ACL looks something like this:**
  - **SourceMAC = AssignedMACAddress; permit**
  - **SourceMAC[47..24] = FC-MAP; deny**
    - (there are other items related to discovery that are not relevant to this case study)
- **The assigned MAC address is discovered via "FIP Snooping" (similar to IGMP snooping)**
  - **The bridge observes the log in responses from the FCF**
  - **When the log in response is observed, the "permit" term is added to the ACL**

ACL Case Study – The Tagged Environment

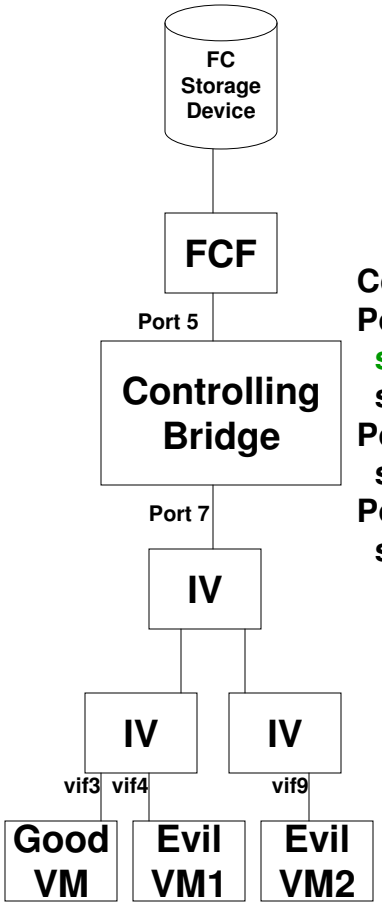


Controlling Bridge ACLs:
Port 7.3:
smac[47..24] = FC-MAP; deny
Port 7.4:
smac[47..24] = FC-MAP; deny
Port 7.9:
smac[47..24] = FC-MAP; deny

- **Initial state before Fibre Channel Login**
- **The good VM will issue an FCoE FIP FLOGI, requesting an FCoE MAC address**
- **The FCF will respond with the FCoE MAC address**
- **The Controlling Bridge snoops the response, looks up the destination port (7.3), and adds the appropriate entry in the ACL**

See next slide

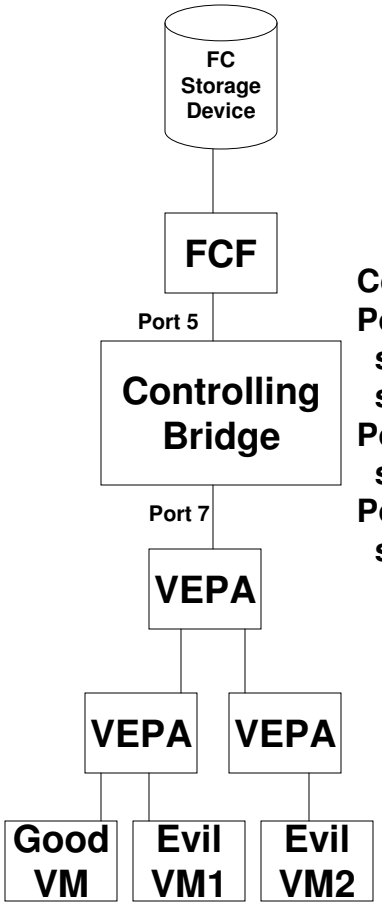
ACL Case Study – The Tagged Environment



Controlling Bridge ACLs:
Port 7.3:
smac = AssignedMAC; permit
smac[47..24] = FC-MAP; deny
Port 7.4:
smac[47..24] = FC-MAP; deny
Port 7.9:
smac[47..24] = FC-MAP; deny

- **This shows the updated ACL**
- **If a frame is received with the AssignedMAC and tag indicating it came from port 7.3, it is permitted**
- **If a frame is received from any other VM with the AssignedMAC (or any FCoE MAC address), it is denied**
- **Thus, the Evil VMs cannot use the MAC address assigned to the good VM**

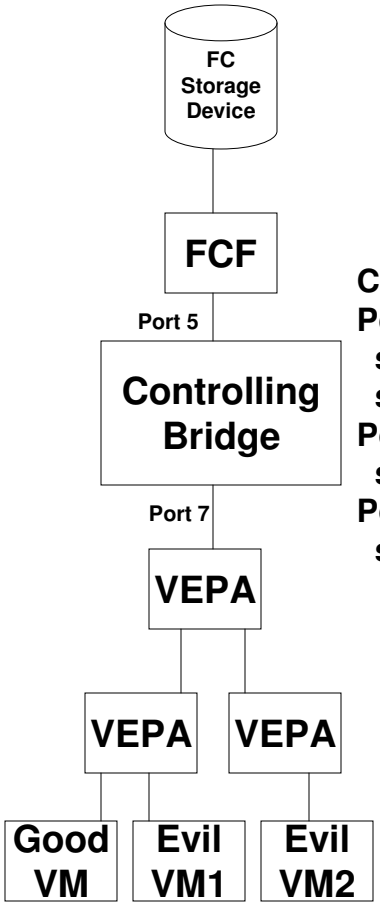
ACL Case Study – The Untagged Dilemma



Controlling Bridge ACLs:
Port 7.idontknow:
smac = AssignedMAC; permit
smac[47..24] = FC-MAP; deny
Port 7.idontknowthiseither:
smac[47..24] = FC-MAP; deny
Port 7.northisone:
smac[47..24] = FC-MAP; deny

- The controlling bridge enforces ACLs
  - The controlling bridge does not have a tag to indicate from which port the frame arrived
  - Therefore, a port specific ACL cannot be constructed
- How about having the VEPA enforce source MAC address?
  - See next slide

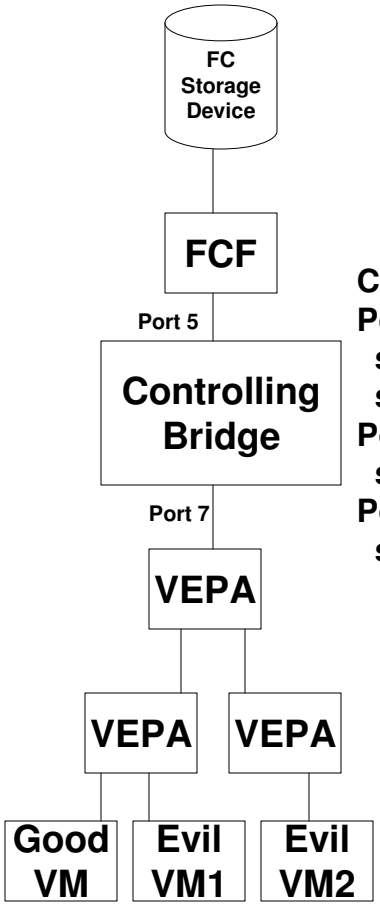
ACL Case Study – The Untagged Dilemma



Controlling Bridge ACLs:
Port 7.idontknow:
smac = AssignedMAC; permit
smac[47..24] = FC-MAP; deny
Port 7.idontknowthiseither:
smac[47..24] = FC-MAP; deny
Port 7.northisone:
smac[47..24] = FC-MAP; deny

- **Ok, the VEPA could enforce Source MAC address, and ACLs could use source MAC address as a "proxy" for the source port number**
- **Not so fast...how does VEPA know the valid addresses for the port?**
  - **The hypervisor tells it. How does the hypervisor know?**
    - It assigns it. BUT, not in this case. The MAC was assigned directly to the VM by the FCF
    - Note: this is not unique to FCoE, the Transparent Services have a similar characteristic
  - **The expectation is that the VM will configure the MAC addresses that it uses and this will trickle down through the hypervisor**
  - **But wait, we have Evil VMs…**
- **The chain of trust if fundamentally broken in this case**
- **But Wait – Maybe the controlling bridge should tell the VEPA (after all, its reasonably trustworthy)**
  - **See next slide**

ACL Case Study – The Untagged Dilemma



Controlling Bridge ACLs:
Port 7.idontknow:
smac = AssignedMAC; permit
smac[47..24] = FC-MAP; deny
Port 7.idontknowthiseither:
smac[47..24] = FC-MAP; deny
Port 7.northisone:
smac[47..24] = FC-MAP; deny

- **Ok, we cannot trust the VMs, since they may be evil. But the controlling bridge does the snooping, so it can tell the VEPA the assigned MAC**
  - **But how? It knows the assigned MAC, but it does not know to which VEPA port it belongs.**
    - VEPAs do not enumerate ports to the controlling bridge
  - **We could identify the VEPA port using a MAC address**
    - But we already determined that VMs must be able to specify their own MAC addresses, so we cannot trust the MAC address as a proxy.
- **We could push ACL processing into the VEPA**
  - **This solves the problem technically. However, ACL processing is very expensive in transistors, power, etc. It probably would be much easier for the VEPA to simply tell the controlling bridge from which port each frame was received, and let the Controlling Bridge deal with the ACLs**

Case Studies

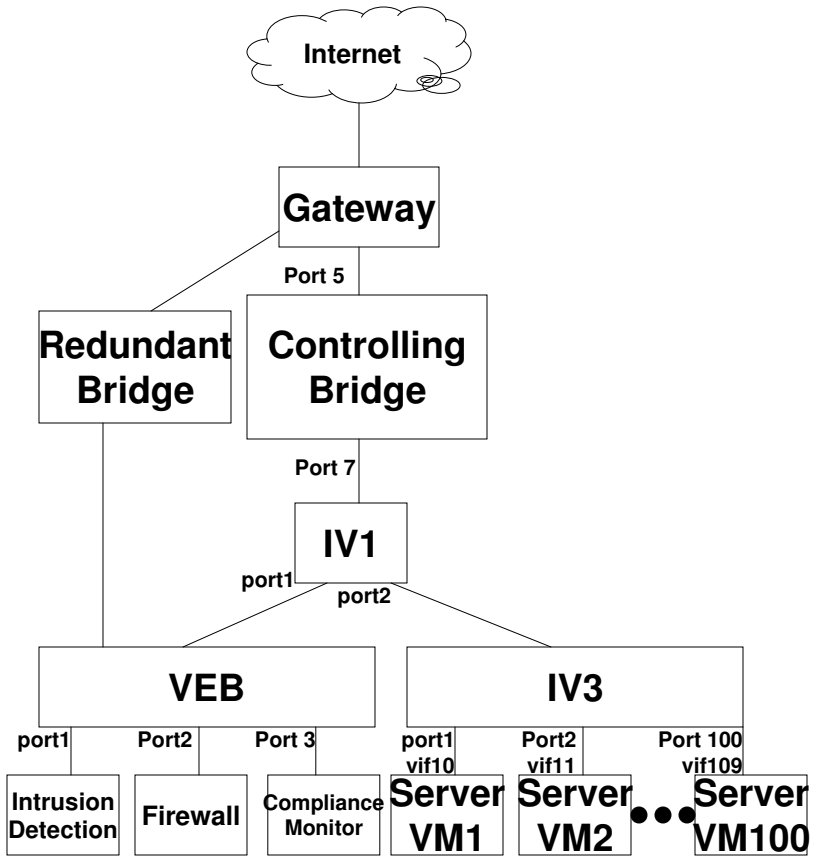
Bridge Stacking:

Stacked Services Example

Bridge Stacking Case Study - Background

- **VNTag (and VEPA) proposes that all VM to VM traffic leaves the server, passes through the controlling bridge, and returns to the server**
  - **In many applications, the amount of VM to VM traffic is small, and therefore the benefits of VNTag far outweigh this traffic flow characteristic**
  - **In other applications, large amounts of VM to VM traffic are present, and therefore a VEB may be a more appropriate choice**
- **This implies that the VNTag devices must co-exist with VEB (and other bridges)**

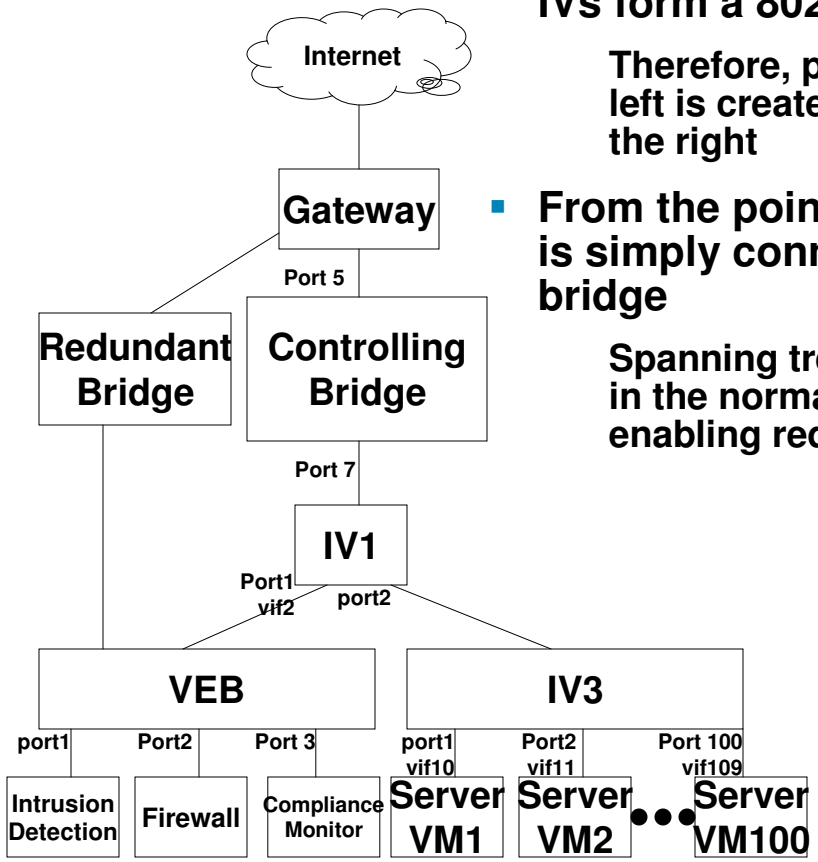
Bridge Stacking Case Study – Stacked Services

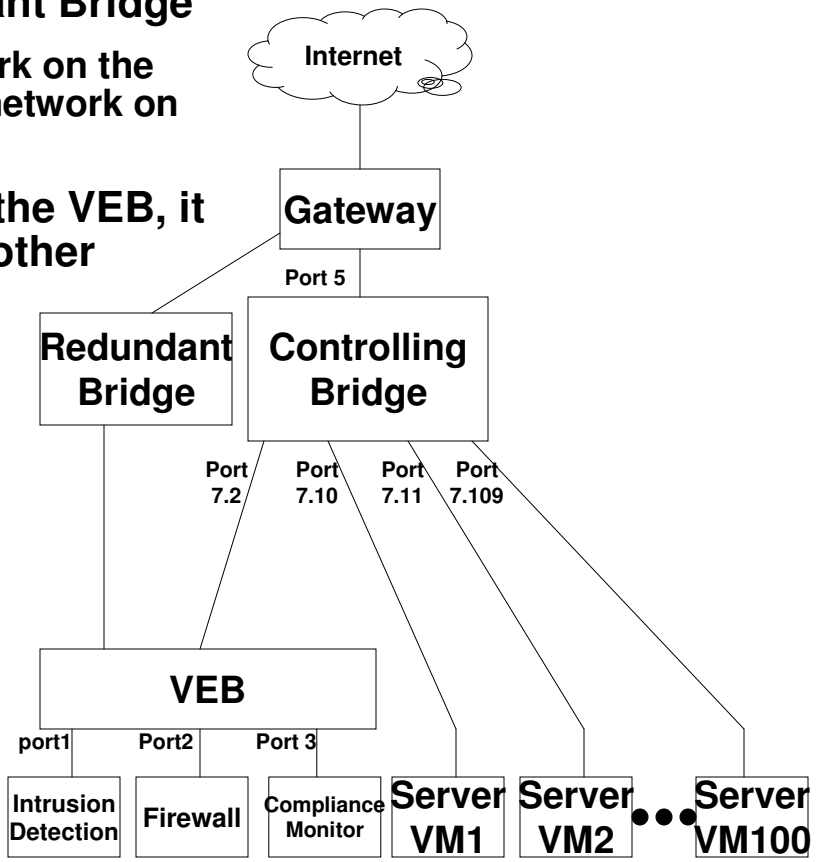


- Transparent Services may be stacked to provide a combination of services
  - In this example, each frame flows through an Intrusion Detection, Firewall, and Compliance Monitor Service
- Clustered services is a similar example
- In this case, there is extensive VM to VM traffic between the services
  - Therefore, a VEB was selected for this server

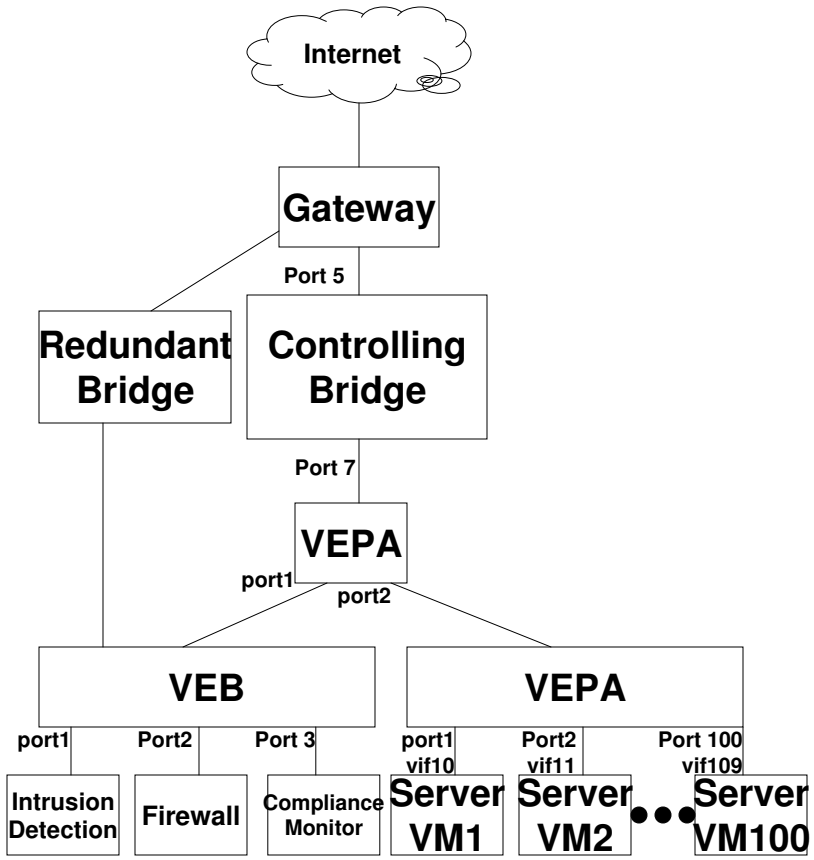
Bridge Stacking Case Study – Stacked Services

- **A Controlling Bridge and its set of IVs form a 802.1Q Compliant Bridge**
  - **Therefore, physical network on the left is creates the logical network on the right**
- **From the point-of-view of the VEB, it is simply connected to another bridge**
  - **Spanning tree operates in the normal fashion enabling redundancy**





Bridge Stacking Case Study – The Untagged Dilemma



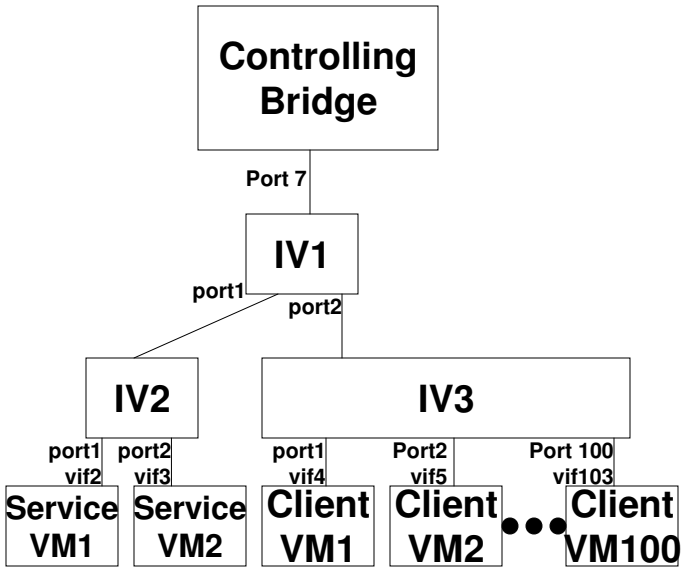
- The concept of attaching anything other than a VNIC or VEPA extender to a VEPA downlink appears to beyond the scope of the VEPA proposal
  - VEPA cannot direct BPDUs to a particular port
  - The VEB would need to speak "VEPA" to program the MAC addresses
  - Hypervisors would need to "reach through" the VEB to program their addresses
  - The controlling bridge would not be able to identify from which port a BPDU was received
- This seems overly restrictive limiting the effectiveness of the new standard
  - If a mechanism is provided to allow the Controlling Bridge to direct a frame to a given VEPA port, and to determine from which port frames are received, these limitations may be removed

Case Studies

Multicast Egress ACLs:

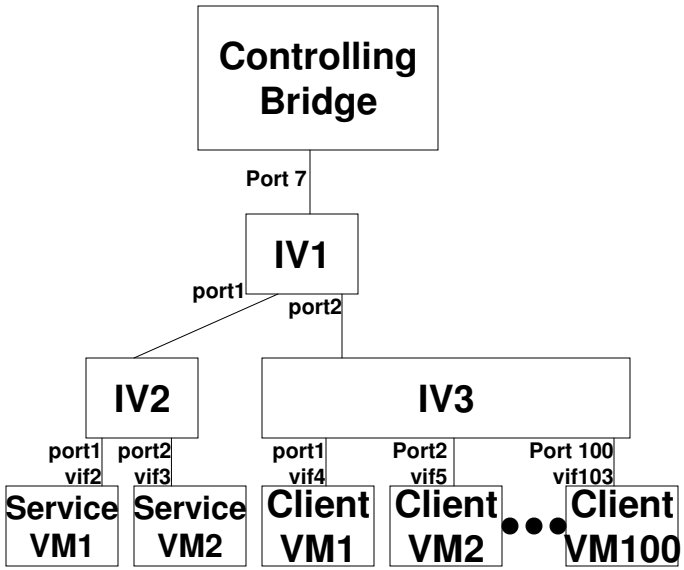
Service Load Balancing Example

Multicast Egress ACL Case Study - Background



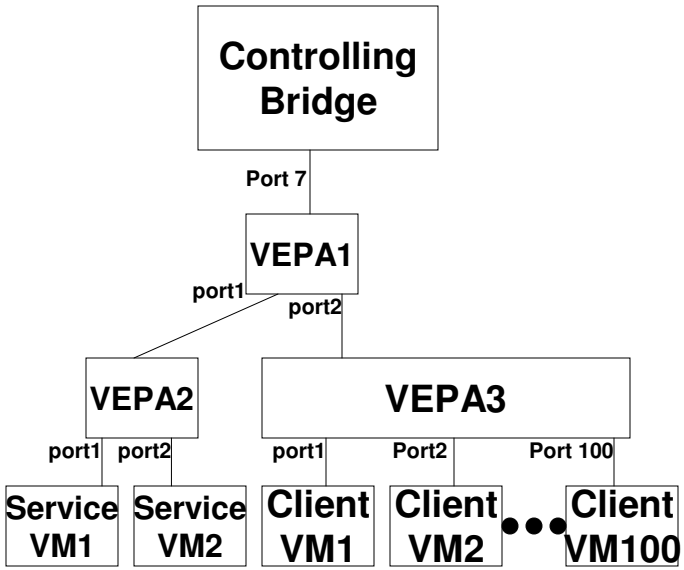
- **Many services advertise their presence via messages sent to a well known group address**
  - **An FCoE FCF is one of many examples**
- **Egress ACLs are an efficient and common way to load balance between multiple instances of a service**

Multicast Egress ACL Case Study – Service Balancing



- **Lets assume that I want SeviceVM1 to serve all even numbered ports and ServiceVM2 to serve all odd numbered ports**
  - **Simply achieved by creating an ACL for each egress port, for example:**
  - **Port 7.4 (and all even numbered ports):**
    - dmac=WKA, smac=ServiceVM1; permit
    - dmac=WKA; deny
  - **For all odd ports:**
    - dmac=WKA, smac=ServiceVM2; permit
    - Dmac=WKA, deny
- **The Controlling bridge achieves this by allocating two multicast vif_list_ids**
  - **One includes all of the odd ports, the other all of the even ports**
  - **The appropriate vif_list_id is chosen based on the result of the egress ACL processing**

Multicast Egress ACL Case Study – The Untagged Dilemma



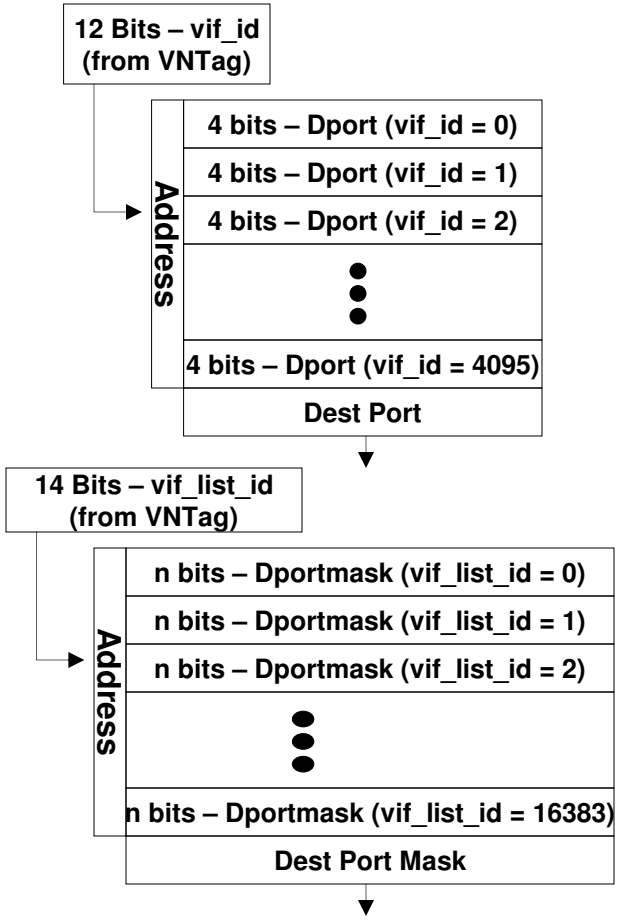
- **VEPAs forward based on destination MAC address**
  - **Which happens to be a multicast address in this case**
- **However, in this case, the controlling bridge cannot do the ACL processing**
  - **It has no way to inform the VEPAs whether the frame is to go to even or odd ports**
  - **All of these frames have the same destination MAC address, so it does not help**
- **The VEPA could do ACL processing**
  - **However, this is very expensive in terms of transistors, power, etc.**
- **Alternatively, the VEPA could allow the Controlling Bridge to due to ACL processing if it would allow the Controlling Bridge to provide an alternate indication of the destination ports**

Case Studies

Comparative Forwarding Logic

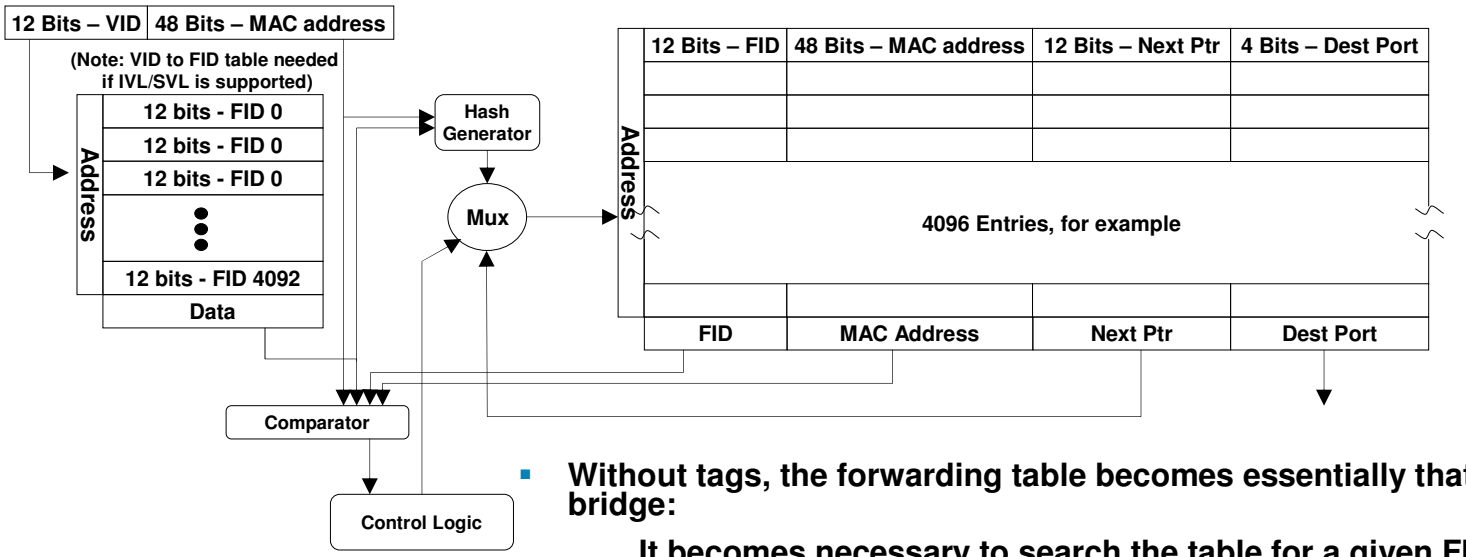
IV Forwarding Tables

Typical IV Implementation



- **The forwarding tables used by Interface Virtualizers are very simple and have many desirable characteristics:**
  - **The tables are directly indexed, vs. requiring the table to be searched**
  - **The entries are small**
  - **A downlink using multiple MAC addresses / VLANs do not consume additional entries – enhanced scalability**
  - **The entries are static; MAC learning / aging do not require changes to the tables**
  - **Supports direction of arbitrarily addressed frames**
    - e.g. egress ACL processing of multicast frames
    - Delivery of PDUs addressed to well known addresses
- **Additional tables not needed (functionality is in Controlling Bridge)**
  - **MAC / VLAN forwarding table**
  - **VLAN member sets**
  - **ACLs (and associated TCAM)**

Forwarding Table – The Untagged Reality



- **Without tags, the forwarding table becomes essentially that required by a bridge:**
  - **It becomes necessary to search the table for a given FID/MAC combination**
  - **Each entry is large (76 bits in this particular example)**
  - **A downlink using multiple MAC addresses / VLANs consume unique entries**
    - 4k Entries may be insufficient
  - **The entries not are static; a new MAC / deleted MAC requires table updates**
    - VM migration delayed / complicated
  - **Does not supports direction of arbitrarily addressed frames**
    - e.g. egress ACL processing of multicast frames
    - Delivery of PDUs addressed to well known addresses
- **Additional (possibly large and complex) tables may be needed:**
  - **VLAN member sets**
  - **ACLs (and associated TCAM)**

Coexistence of VEB and IV / VEPA

Coexistence of VEB and IV / VEPA

- There seems to be consensus that VEBs are an important part of the data center environment
- VEPA appears to contain most of the complexity of a bridge:
  - Needs to do full VLAN/MAC address forwarding
  - Needs full ACL processing
  - Needs full VLAN functionality
  - Needs to learn/age
- There appears to be little point in developing a VEPA only device
  - VEPA seems to be a special operating mode of an embedded bridge

Coexistence of VEB and IV / VEPA

- **An Interface Virtualizer is far less complex than a VEB or VEPA:**

  Most of the complex functions are allocated to the Controlling Bridge:

  VLAN/MAC address forwarding

  ACL processing

  VLAN functionality

  Learning / aging

- **Thus an IV is a very simple device and may stand alone.**
- **However, it is also desirable to have combined IV/VEB functionality**

Coexistence of VEB and IV / VEPA

- One could argue that since VEPA is nearly a bridge anyway, that starting from a VEB, VEPA is easier to build

  Does this make sense to do?

  When does a VEPA make sense to deploy over a bridge?

  Since the VEPA is performing most of the bridge functionality anyway, is there any efficiency to be gained on operational costs or ease of management, that could not equally be applied to a VEB?

Coexistence of VEB and IV / VEPA

- **A combined VEB / Interface Virtualizer is a very interesting device:**

  Independent operating modes: allows administrator to select mode of operation to fit current VM operational characteristics:

  e.g. VEB for high VM to VM traffic

  IV for mainly VM to external traffic

  Gain operational and administrative efficiency by eliminating a the internal bridge

Coexistence of VEB and IV / VEPA

- **Very cool capability: operation as a VEB and IV simultaneously**
- **Greatly simplifies "feature creep" demands on VEB**
  - **Optimize for trusted VM to VM traffic**
  - **Add in any value add desired**
  - **Use IV functionality for everything else**
- **You get all this just by adding a tag!**

A Call for Interest

Embedded Bridge / VNTag hybrid operation

Call for interest

- It seems clear (at least to me ☺) that we need:
  - Embedded bridging (both hardware and software based)
  - Port Extension
- However, it would also be interesting to explore a hybrid mode of operation (i.e. both modes operating simultaneously):
  - Fast path bridging for applications with high VM to VM traffic
  - IV functionality for traffic requiring the advanced bridging capability of the Controlling Bridge
  - A simple protocol to coordinate this between the controlling bridge and the hybrid device
- If you would be interested in exploring such a capability, please contact me (jopeliss@cisco.com).
  - Hopefully there will be sufficient interest and we can get an informal study group together

Summary

Summary

- **The VNTag approach provides a clean, straight forward, and complete approach to address the problems associated with bridge proliferation in modern data center environments**
  - **Provides a simple, low-cost alternative that can dramatically reduce the number of bridges**
  - **Interoperates with independent bridges, including VEBs, to support the applications where they are needed**
  - **Continues to provide (and in many cases enhance) data center critical capabilities**
    - Non-hypervisor assigned addresses, VLAN enforcement, ACLs, rapid VM migration
  - **Addresses data center "pain points" beyond just the bridge embedded in a virtualized server in a logical and consistent manner**

Questions?

Thank You!