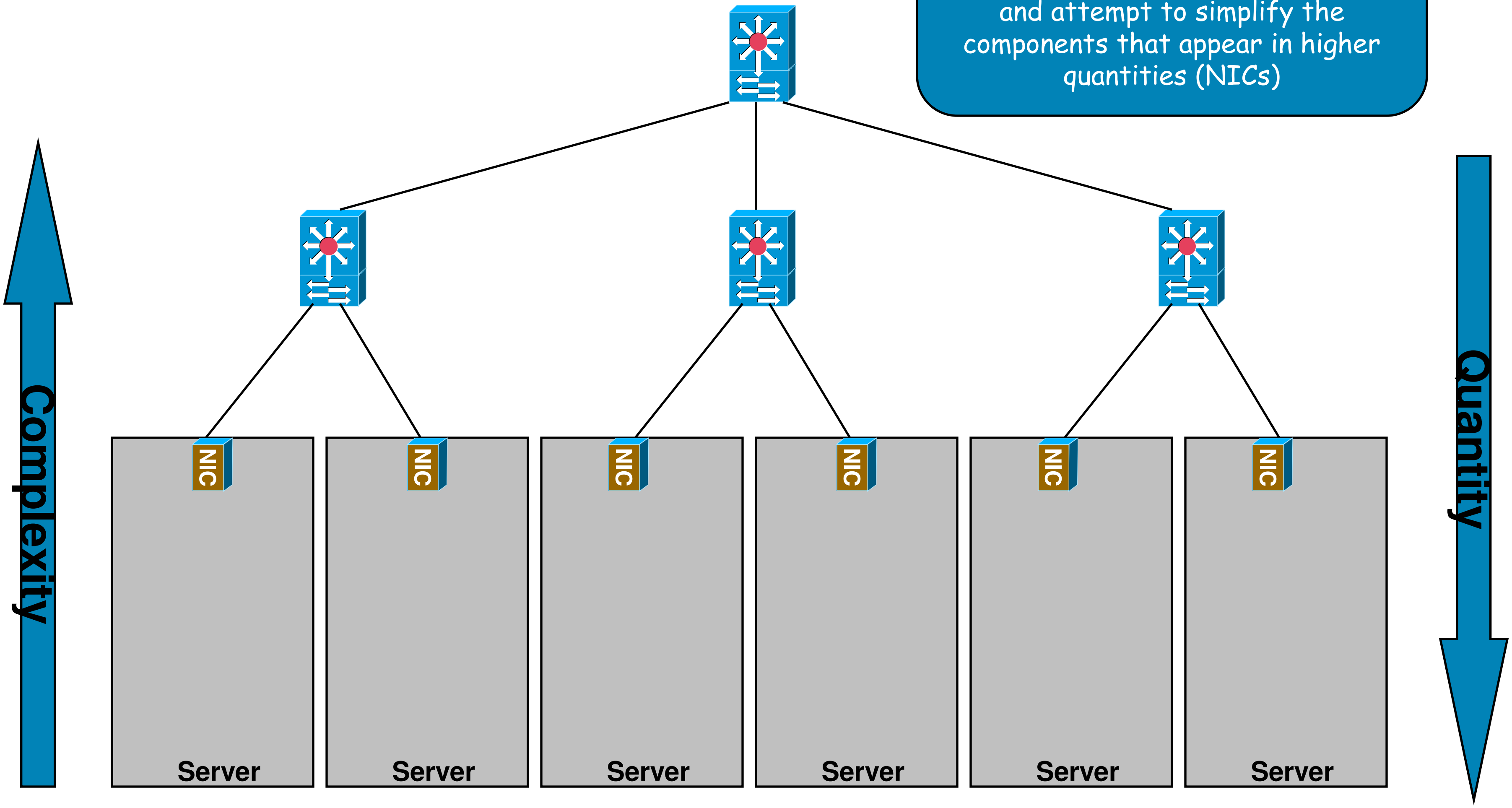
Network Interface Virtualization Review

Joe Pelissier

new-dcb-pelissier-NIV-Review-0109

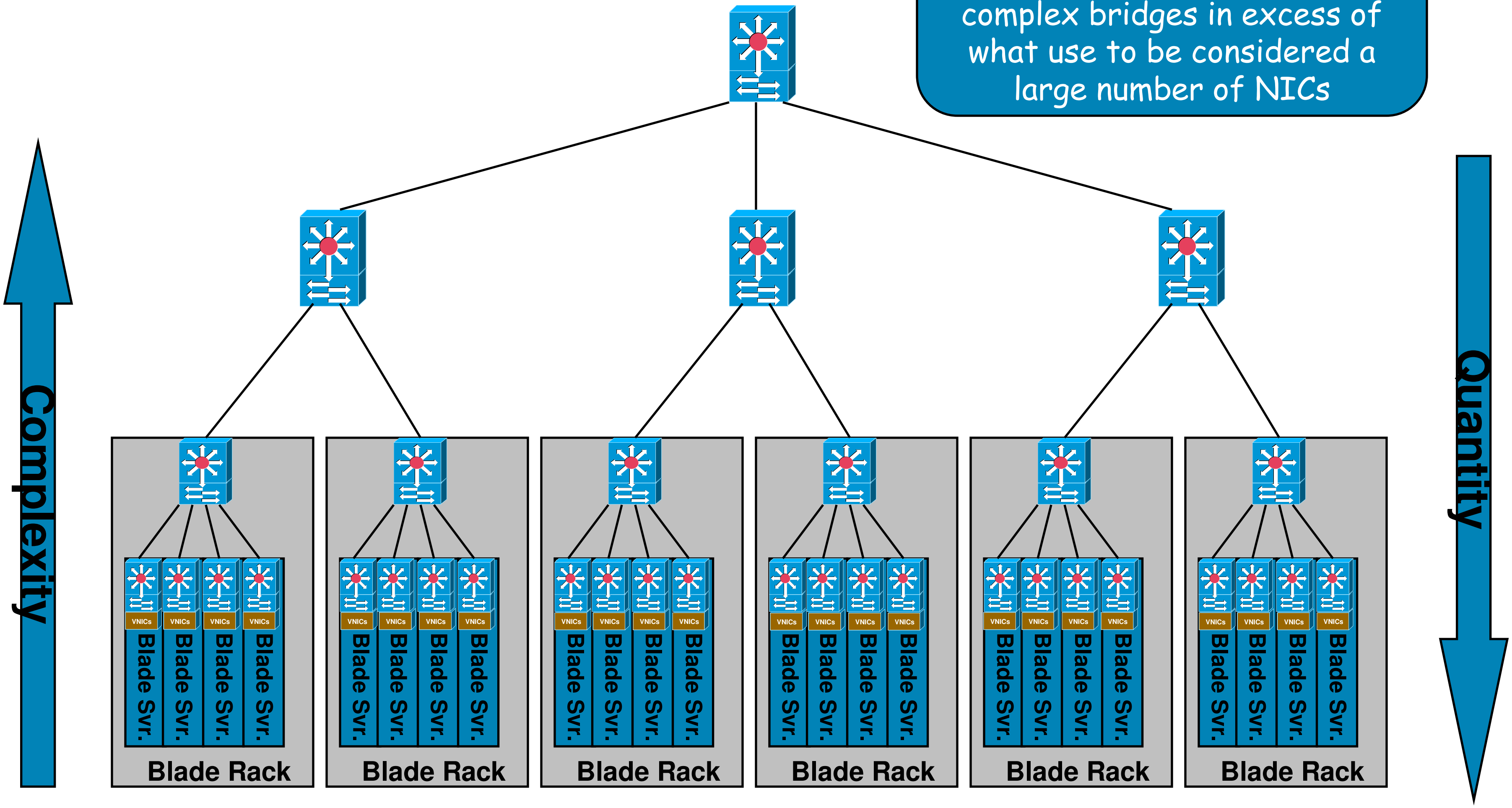
Motivation

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)



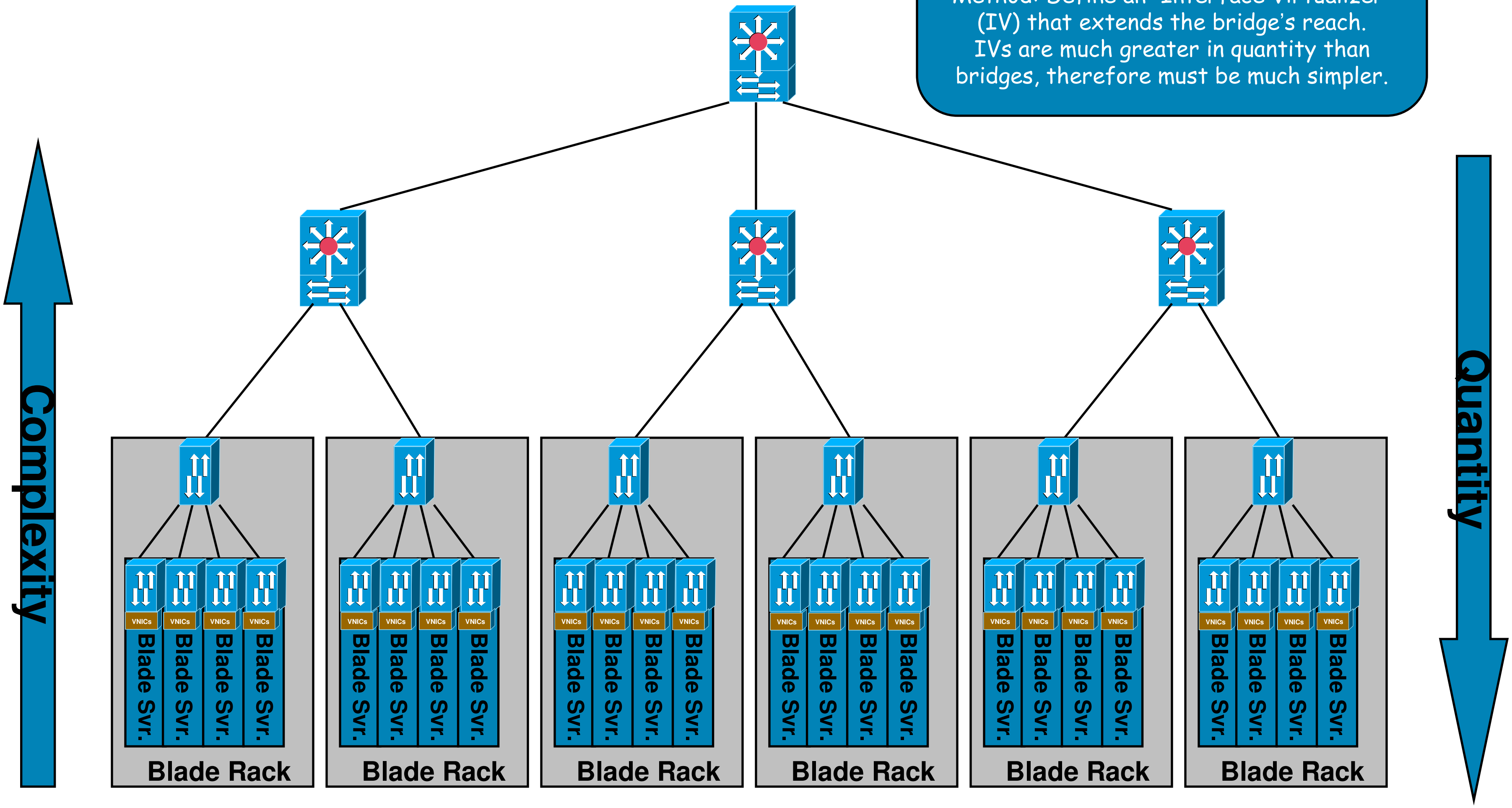
Motivation

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



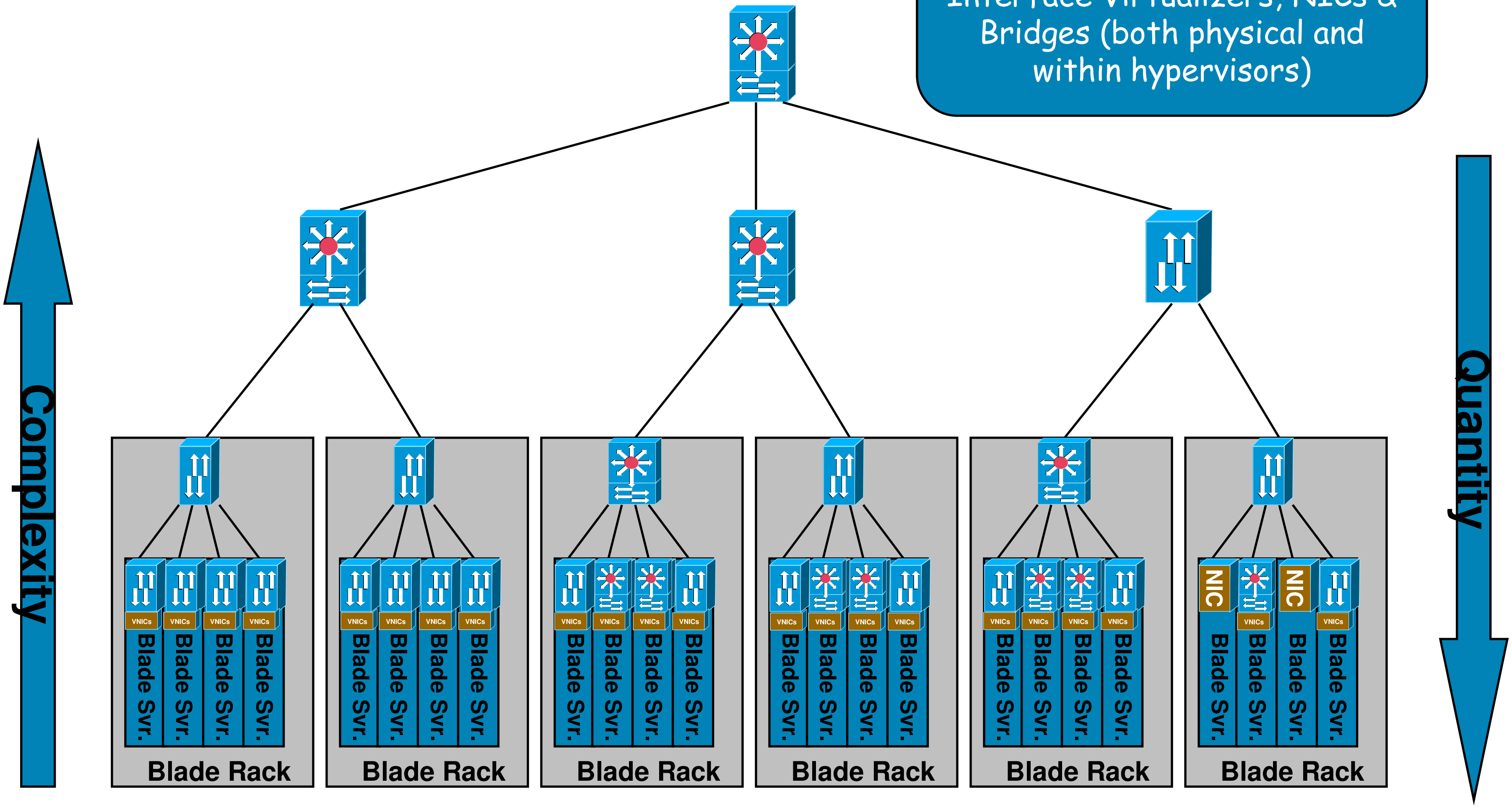
Motivation

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

Evolutionary deployment will require support of a mix of Interface Virtualizers, NICs & Bridges (both physical and within hypervisors)



Requirements Summary

- **Must be simple**

  Drive complexity towards the bridge and simplicity towards the NIC

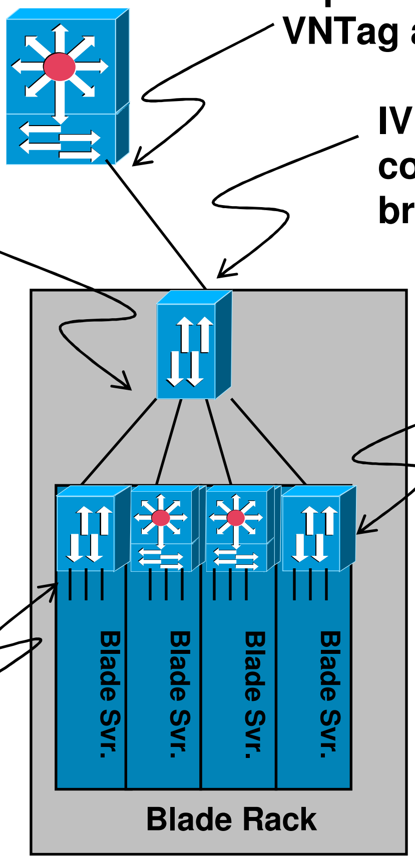
  For example, ACL processing, CAM lookups, learning and aging functions, etc.

- **Must operate in a variety of configurations**

  Downlinks may be connected to other Interface Virtualizers, bridges, or NICs

  These devices may be virtual, instantiated together, or physically separate

Anatomy of an IV fabric



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

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

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

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.

Interface Virtualizer Basic Functions

- **From NIC to Bridge**

  Add VNTag on ingress (indicating source IV port)

  The Endstation does not add VNTags nor is it required to have any "VN awareness" -Completely backwards compatible

  Forward frame up the IV hierarchy to the bridge

- **From Bridge to NIC**

  Froward frame down hierarchy to the NIC

  Based on tag information

  Replicate multicast frames

  Filter the frame at the ingress port if it was sourced at the IV

  Remove the VNTag at the final IV

From the "outside world", the collection of an IV capable bridge and its IVs appears as a single bridge. Other devices connecting to this combination require no "IV" awareness

Goals of the VNTag

- For frames from the bridge to the VNIC, the tag should provide a simple indication of the path through the IV(s) to the final VNIC.
- For frames from the VNIC to the bridge, the tag should provide a simple indication of the ingress port of the southern most IV.
- For multicast frames originating from somewhere else in the network, provide a simple pointer to a "replication table" within the IV.
- For multicast frames originating from one of the VNICs, provide #3 plus an indication of the source VNIC to prevent replication of the frame back to the source.

Virtual Interface Identifiers

- **Each downlink from an IV to a VNIC is, in effect, a bridge interface**
  - **These are the physical instantiations of virtual interfaces on the bridge itself**
  - **Each is identified by a 12-bit Virtual Interface Identifier (vif_id)**
    - Assigned by the bridge to each IV downlink port
- **In addition, each IV may be programmed with lists of downlink ports (for use in multicast)**
  - **Lists are identified by a 14-bit vif_list_id**

VNTag Proposal

Ethertype				d	p	Dvif_id or vif_list_id
l	r	ver	Svif_id			

Ethertype: TBD, identifies the VNTag

d: Direction, 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 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: Pointer to a list of downlink ports to which this frame is to be forwarded (replicated)

Dvif_id: Destination vif_id 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 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: reserved

ver: Version of this tag, set to 0

Svif_id The vif_id 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.

Interface Virtualizer Operation

- **From Northbound fames (Downlink to Uplink, d=0)**
  - **If no VNTag present, add one**
    - Set Svif_id to vif_id of ingress port, all other fields set to 0
  - **Forward to uplink**
    - Support of multiple uplinks to be discussed later

Interface Virtualizer Operation

- For Southbound frames (Bridge or uplink to downlink)
  - If unicast: forward to downlink port corresponding to Dvif_id
  - If multicast: forward to set of downlink ports indicated by vif_list_id
  - If the downlink port's vif = the frame's Svif_id, filter
  - If the downlink is not known to be connected to another IV, remove the VNTag

Note that the size of the vif_id is intentionally chosen to be small enough to use as an index into a forwarding table: No table searches are required.

Bridge use of VN_Tag

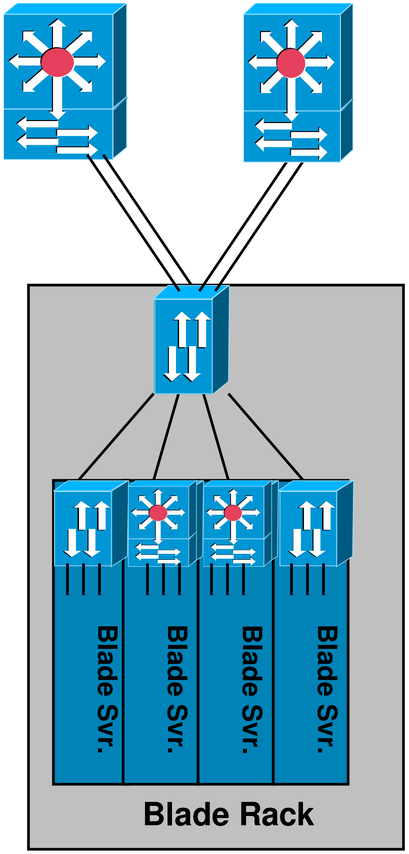
- On ingress
  - Learn MAC address to vif_id as part of normal bridge learning function
- On egress: set VNTag as follows:
  - Set the Dvif_id based on the MAC Address

Forwarding Tables

- **VIF forwarding table**
  - **One entry per VIF_ID**
    - May support up to 1024 unique VIFs
    - Indexed by Dvif_id
    - Entry points to downlink to be used
- **VIF list table**
  - **One entry per vif_list_id**
    - May support up to 4098 unique lists
    - Indexed by vif_list_id
    - Bit mask indicating which downlinks are to be used
      - Width of entry depends on number of downlink ports

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



Virtual Interface Control (VIC) Protocol

- Bridge configures all of the forwarding tables for each downstream (i.e. cascaded) IV
- VIC Protocol provides this functionality
  - Independent instance of VIC is executed for each Uplink Port (or Uplink Port Aggregation)
- No frames may flow through an IV until the IV is configured by the VIC protocol
- VIC Protocol operates between the bridge and the VIC controller in each IV
  - A MACSEC SA may be established between the bridge and each VIC Controller to secure the VIC Portocol
  - VIC Controller acts as an endstation to the bridge to establish the SA

MACSEC and VNTag

- The collection of IVs and the IV Capable bridge operate as a single bridge
- An endstation wishing to execute MACSEC operates identically as if it were connected directly to a bridge
- Within the “IV Cloud”, the SA is actually established between the IV capable bridge and the endstation
  - The IVs pass the secured frames, including the SecTAG transparently between the endstation and the IV capable bridge
  - Much like a provider network would do…

Uh…but wait a minute…

- **Doesn't that mean that the VNTag is not protected by MACSEC?**
  - **Yes, again it works much like it would in a provider network**
- **Does this cause a security risk?**
  - **No, and here's why:**
  - **We need consider only an attack on an IV to IV or IV to IV capable bridge link (these are the only ones that have VNTags)**
  - **If the VNTag is modified and the MAC address is not**
    - The frame is misdelivered (which can occur anyway)
    - Misdelivery detected by mismatch of MAC address
  - **If the VNTag is modified and the MAC address is correspondingly modified**
    - MACSEC detects the corruption of the MAC address

Can we use the SecTAG?

- **It has been proposed that the SecTAG could be used to carry the information in VNTag and thus eliminate the need for a new tag**

  Note that these functions operate independently:

  One may choose to use virtualization alone (the most common case), MACSEC alone, or both together

  All of these cases must be reasonably handled

SecTAG for Virtualization

- **Paul Congdon and I have worked together to develop what we consider is the most optimum approach using SecTAG**

  We disagree on whether using SecTAG provides a better solution than having an independent VNTag ☺

- **The main adaptation of SecTAG is to include the Dvif_ID/vif_list_id and Svif_ID**

  In other words, essential fields of VNTag (unrelated to security) hitch a ride in the SecTAG

  There are a number of secondary adaptations that I will address in a bit…

Adapting the SecTAG

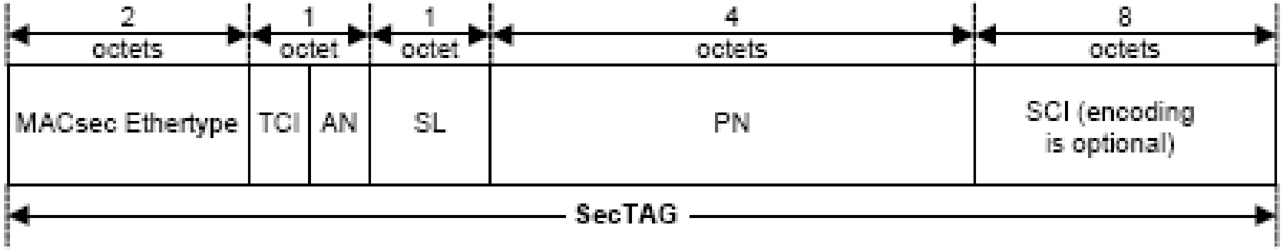


Figure 9-2—SecTAG format

- **Note that the SecTAG contains a 64-bit SCI that contains:**

  A 48 bit MAC address

  A 16 bit virtual port number

- **The SCI identifies the security association**

  Must be unique between all SAs on a given bridge port

  The endstation allocates a port number per SA it creates

  The MAC address ensures SA indentity uniqueness across all devices connected to the bridge

Keeping the SCI unique

- **Note that the MAC address is globally unique and therefore ensures SA identity uniqueness**
  - **However, global uniqueness is not required, just uniqueness across the bridge is required**
  - **Remember that each NIV port is assigned a vif_id, that happens to be unique for a given bridge port**
  - **If communicated to the end station, this may be used in the SCI instead of the MAC address (or at least the OUI part of it)**
    - It’s a lot smaller thus freeing up other bits in the SCI

The New SCI:

- Eliminate the OUI portion of the MAC address (frees 24 bits)
- Keep the virtual port number (16 bits)
- Add the Dvif_id/vif_list_id (reduce to 12 bits)
- Add the Svif_id (12 bits)
- Total: 64 bits
- Note: this eliminates several control bits that are in VNTag

  We can live without them (left as an exercise to the reader)
- Increment the two bit version field to indicate the presence of this new SecTAG format

Endstation use of the SecTAG / VNTag

- **SecTAG case:**
  - **If an endstation is doing MACSEC and virtualization, execute a yet to be defined protocol to discover the vif_id of the IV port to which you are attached (probably just a new field in the MACSEC negotiation).**
  - **If an endstation is doing neither virtualization nor MACSEC, it adds no tag**
  - **If an endstation is doing MACSEC but not virtualization, it uses the old version of the tag (it needs to do this since it will not have a vif_id for uniqueness)**
    - An endstation that wishes to support virtualization must support both forms of the SecTAG
    - Possibly breaks existing implementations / designs in flight.
  - **If an endstation is doing virtualization but not MACSEC, do not include a SecTAG (the IV will do it for you)**
  - **If an endstation is doing virtualization and MACSEC, include the new form of the SECTag**

Endstation use of the SecTAG / VNTag

- **Compatibility alert for the SecTAG case:**
  - **Use of the SCI is currently prohibited for an endstation**
    - At least this is the case if you set the ES bit
    - For virtualization it now becomes required
  - **The old tag format could simply use the same MAC address in the frame header and SCI**
    - The new format no longer does this and requires additional information to be included
  - **The version field will change**
    - Both versions may need to be supported

Endstation use of the SecTAG / VNTag

- **VNTag case:**

  None. Endstations have no awareness of the presence of an IV; it just looks like a bridge port

  Of course, an endstation implementation may elect to embed IV functionality, but the model remains the same

  If the endstation doing MACSEC, include the current version of the SecTAG, otherwise no SecTAG.

  Never add a VNTag (the IV always takes care of)

Endstation Backwards Compatibility

- **SecTAG case:**
  - **An endstation that wishes to use MACSEC in an virtualized environment must be "virtualization aware"**
    - It must be capable of generating the new form of SecTAG
  - **An endstation that wishes to be "virtualization capable" probably must support both versions of the SecTAG**
    - May be deployed in a non-virtualized environment with legacy bridges that do not understand the new SecTAG
  - **Probably requires a discovery mechanism in MACSEC to determine which version of the SecTAG is supported and which should be used**
- **VNTag case:**
  - **Endstations are not required to have any "virtualization awareness" and the SecTAG does not change**

Bridge Backwards Compatibility

- **SecTAG approach:**

  Probably will be required to support both versions of the SECTag

  True even if the bridge does not intend to support virtualization

  The new version is required to claim compliance with the latest standard, the old version is required for connectivity to legacy devices.

- **VNTag approach**

  No changes to SECTag

  Bridges that do not support virtualization interoperate in IV environments without any virtualization awareness

  Fully Interoperable

IV use of the SecTAG / VNTag

- **SecTAG case:**
  - **If the endstation is using MACSEC with the old tag format**
    - Connectivity fails, there is insufficent data for the bridge to learn the ingress IV port
  - **If the endstation is using MACSEC with the new tag format**
    - Verify correct Svif_id is included in the SecTAG
    - Do not add any additional tags
  - **If the endstation is not using MACSEC**
    - Add a SecTAG with the Svif_id
- **VNTag case:**
  - **Add the VNTag**

The MACSEC cipher

- **SecTAG approach:**
  - **If virtualization is being used with MACSEC, then the SecTAG approach requires a method to express a “null cipher”**
    - This would be new to the SecTAG format
  - **Implementation of a “null cipher” implies that the Integrity Check Value (a 16 octet field) becomes optional**
    - This would need to be specified and new implementations would need to accommodate this
    - Probably a compatibility issue for existing MACSEC implementations / designs in flight
- **VNTag approach:**
  - **No impact**

The SecTAG SL field

- **SecTAG approach:**
  - **The SecTAG contains an SL field that contains the length of the frame if it is less than 48 octets**
  - **This has no value for a virtualization only application**
  - **However, it is probably not worth making it optional**
  - **Therefore, an IV will need to calculate this and insert it in the SECTags it creates**
    - Bit of a pipeline headache in the ASIC since you need to buffer a part of the frame before you can create the SECTag header
- **VNTag approach**
  - **The IV does not need to deal with SL**
  - **All of the data required to construct the VNTag is known before a frame arrives**

The SecTAG PN Field

- **SecTAG approach:**
  - **The SecTAG contains a packet number field that increments for each packet transmitted within an SA**
  - **This serves no purpose in a virtualization only application**
  - **We would need to specify that when the null cipher is in use, this field is either omitted, reserved, or at least ignored.**
    - Possible compatibility issue with existing MACSEC implementations / designs in flight
- **VNTag approach:**
  - **No impact**

Architectural Interdependencies

- **SecTAG approach**

  Generates a dependency between the functions that implement MACSEC and those that implement virtualization

  These are frequently in difference chips

  Often on different line cards

  Frequently developed by engineers with significantly different skill sets

- **VNTag approach**

  No significant dependency

Specification Interdependencies

- **SecTAG approach**

  Generates a dependency between the virtualization specification development and the security specifications

- **VNTag approach**

  No significant dependency

- **Once we create this dependency, we live with it forever**

  Already making a significant addressing compromise

  16k multicast + 4k unicast reduced to 4k total just to cram it into the SecTAG

  The natural evolution of these independent technologies will forever require coordination between the two

Deployment Interdependencies

- MACSEC and virtualization are independent from a deployment perspective

  In data centers, it seems likely that the most common deployment will be virtualization without MACSEC

  Outside data centers, it seems likely that MACSEC will be more commonly deployed than virtualization

  And, of course, there will be deployments of both

- SecTAG approach:

  Deployment of MACSEC impacts deployment of virtualization, and vice versa

- VNTag approach:

  Deployment of MACSEC and virtualization are independent

Observation

- **It seems that we are paying a pretty high cost to optimize the case of simultaneous deployment of MACSEC and virtualization**

  Arguably, of the four combinations, this is will be the least prevalent

- **VNTag alone is more optimal for a virtualization only deployment**
- **Having one form of SecTAG is more optimal for a MACSEC deployments**
- **Having both tags really is not that big a deal if you want both of these functions**

  If multiple tags is a concern, we could expand the scope of this effort to include VLAN, priority, and CN indications in the SecTAG ;-)

Thank You!