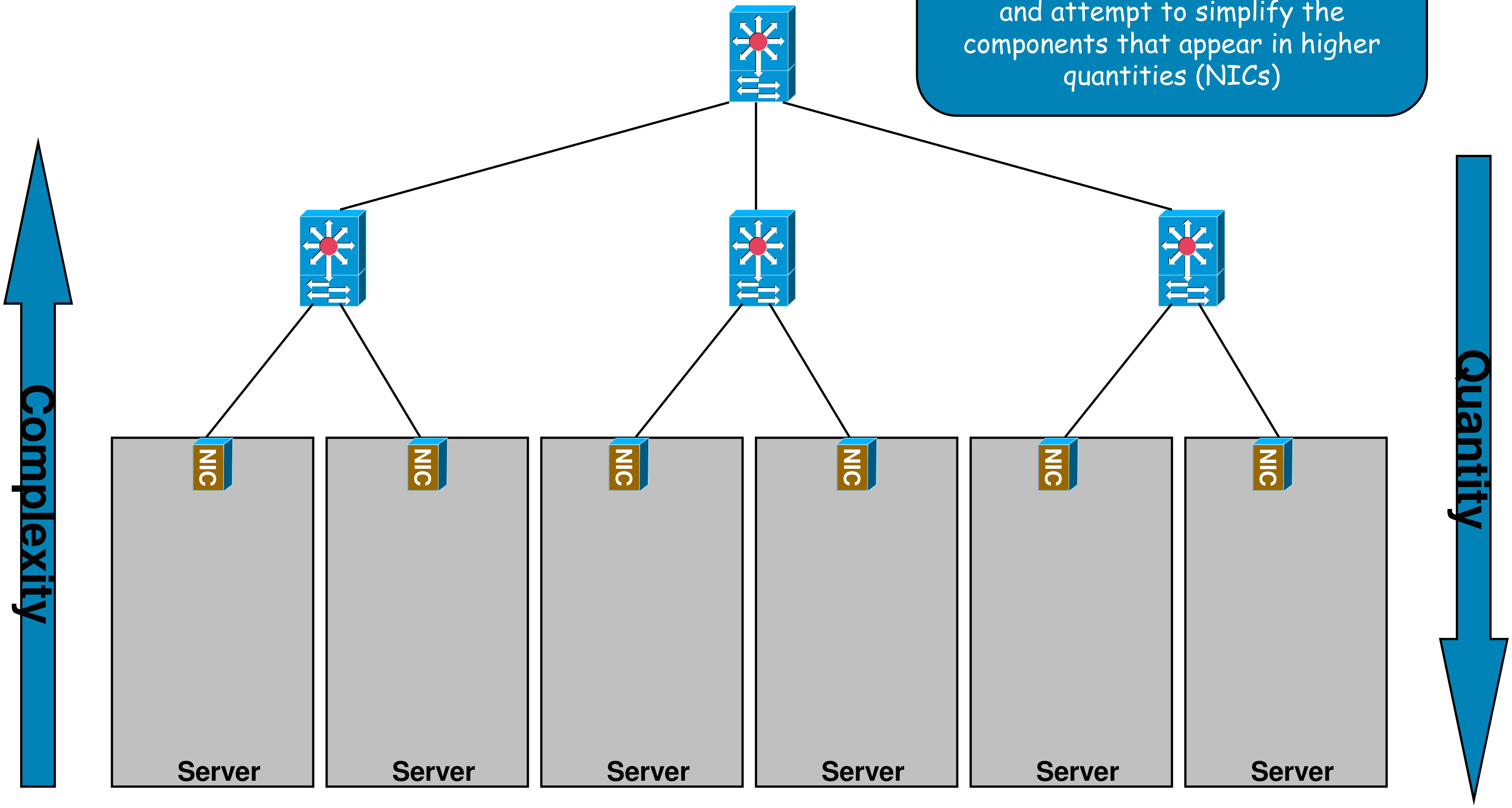
Network Interface Virtualization Proposal

Joe Pelissier

new-dcb-pelissier-NIV-Proposal-1108

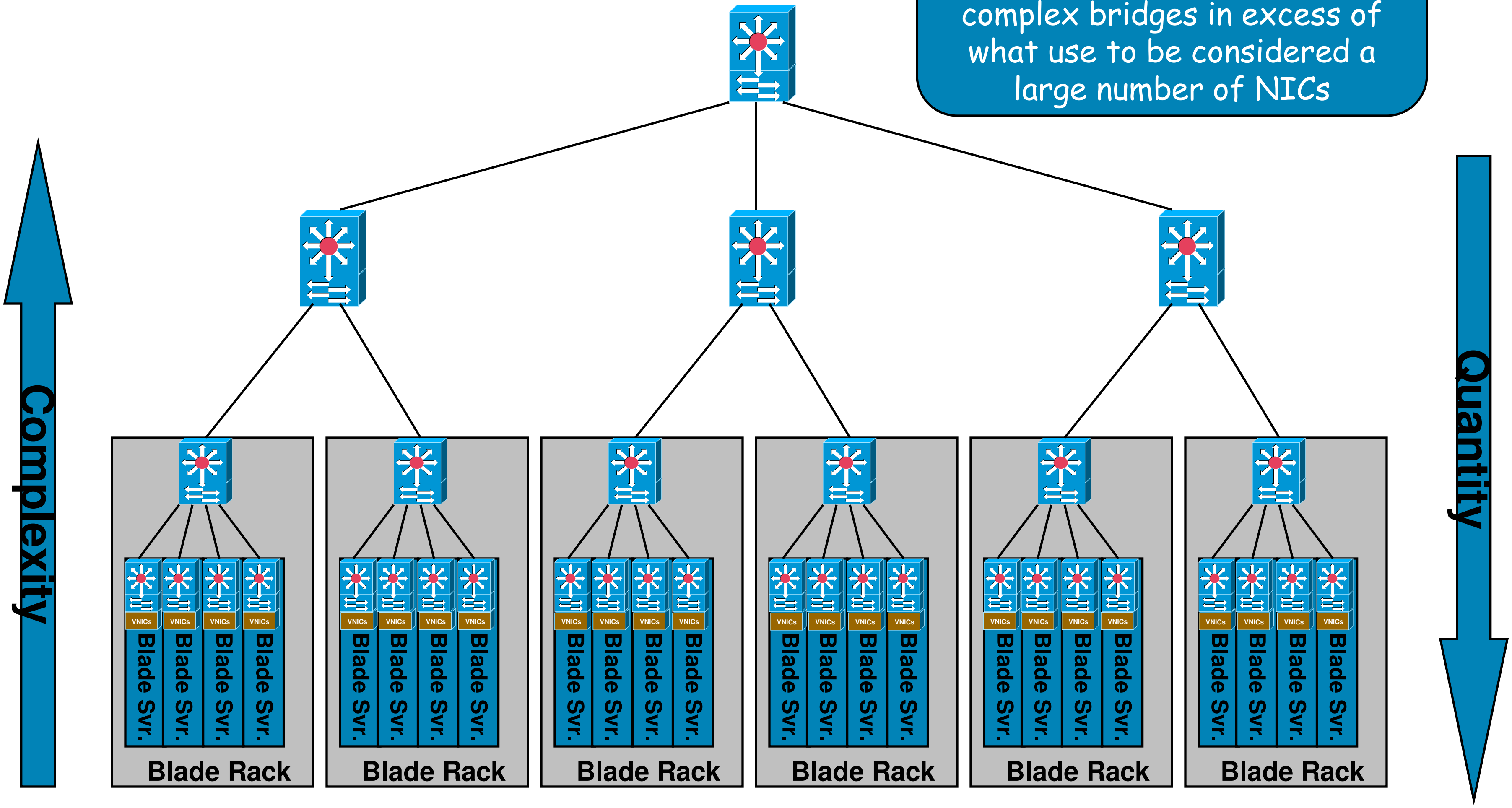
Motivation



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)

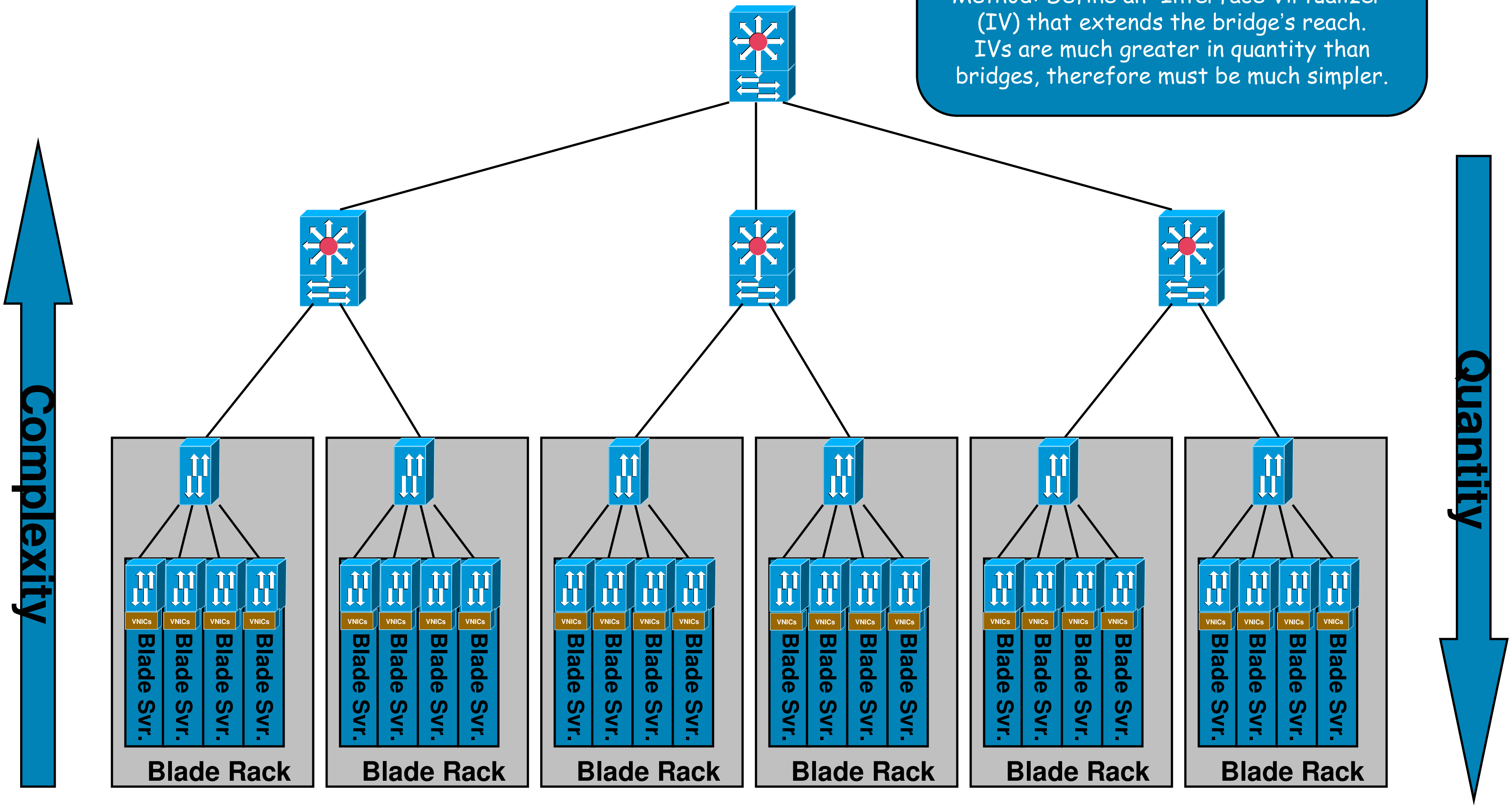
Motivation

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



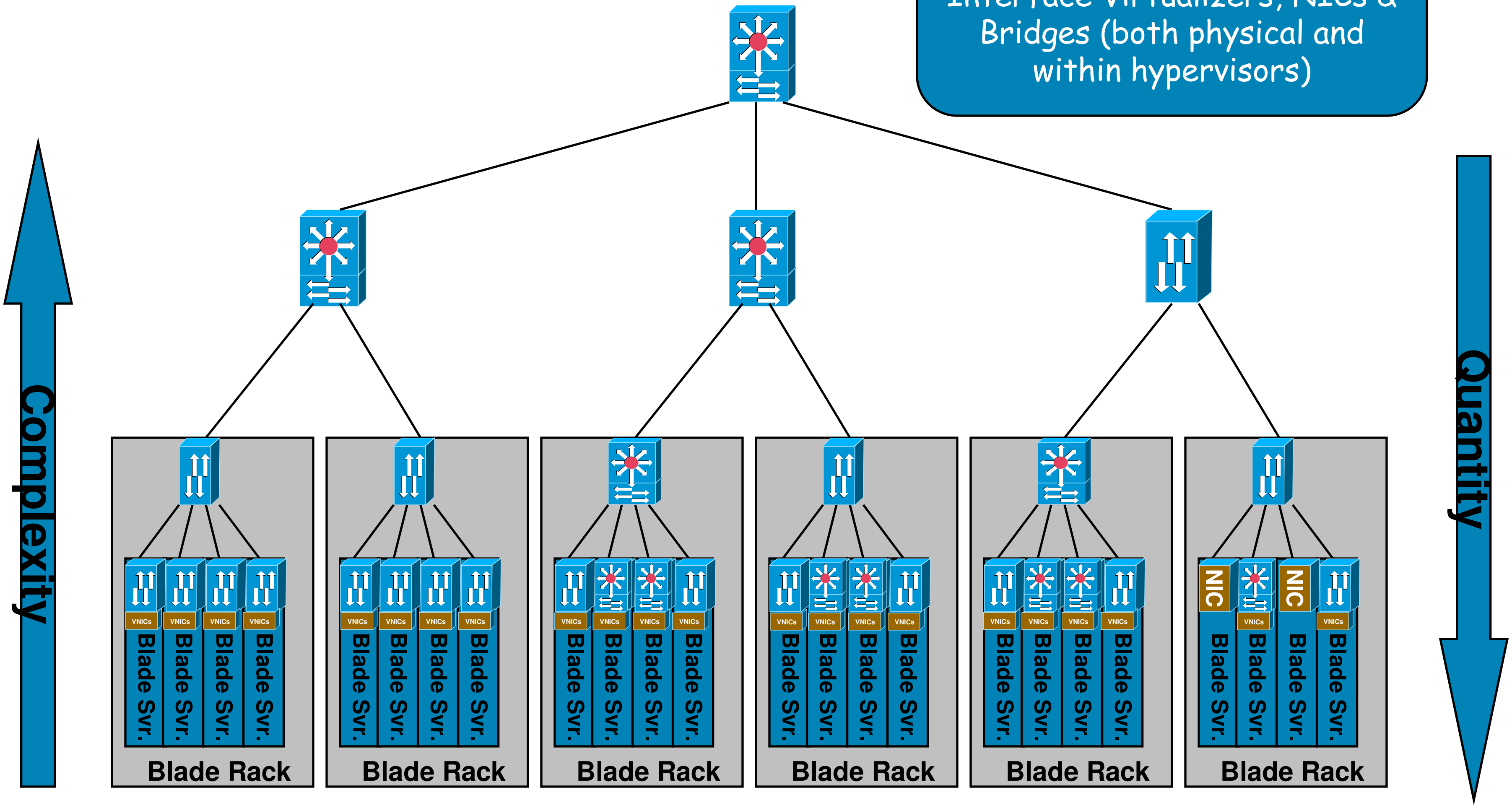
Motivation

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

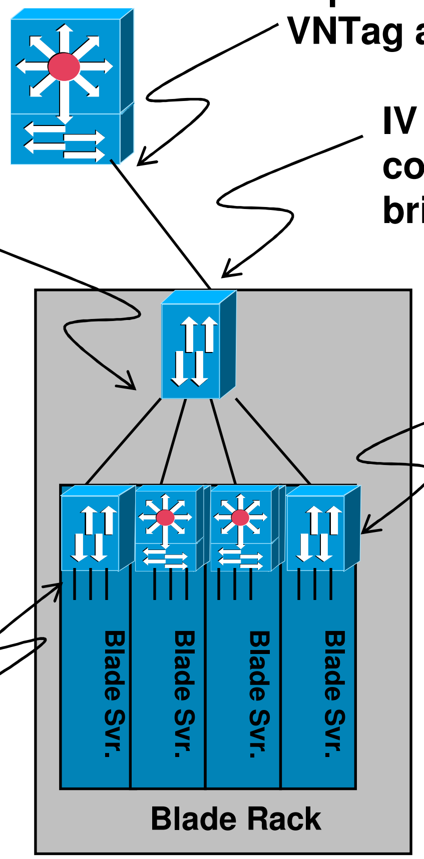
Evolutionary deployment will require support of a mix of Interface Virtualizers, NICs & Bridges (both physical and within hypervisors)



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 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 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.

Interface Virtualizer Basic Functions

- **From NIC to Bridge**
  - **Add VNTag on ingress (indicating source IV port)**
  - **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**

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 source VNIC.
- 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 Downlink to Uplink (d=0)**
  - **If downlink not connected to an IV: Add VNTag**
    - 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

- From Bridge to Downlink (d=1)

  If p=0: forward to downlink ports corresponding to Dvif_id

  If p=1: forward to set of downlink ports indicated by vif_list_id

  If l=1: filter frame if downlink port is connected to a VNIC and its vif = Svif_id

  If downlink not connected to another IV, remove VNTag

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:

  d=1

  l=1 if bridge forwarded the frame on the same physical port on which it was received (e.g. multicast or broadcast), 0 otherwise

  p=0 if frame is to be forwarded to a single IV port, 1 otherwise

  Dvif_id (p=0) set to the vif_id of the egress IV port

  vif_list_id (p=1) set to the vif_list_id of the set of IV egress ports to which the frame is to be delivered

  Svif_id: if l=1, set to the Svif_id included in the frame as it was recevied, 0 otherwise

  All others: set to 0

Additional Interface Virtualizer Functions

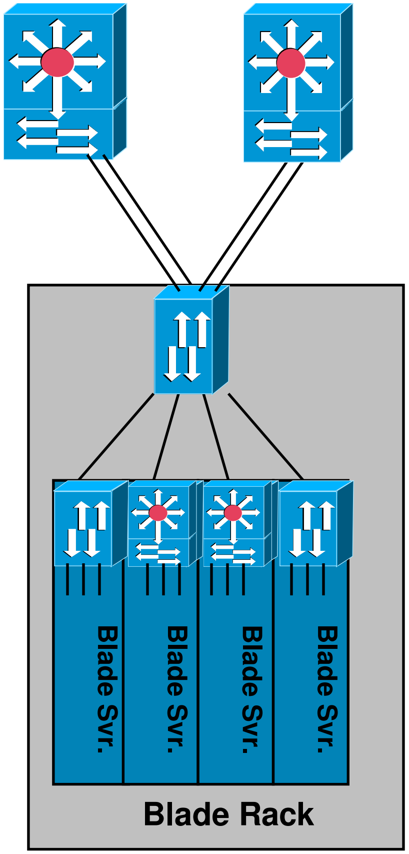
- **Flow control: PAUSE and/or Priority Flow control**
- **Scheduling: strict priority and/or ETS**
- **Frame lifetime: same as 802.1Q**

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**
  - **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)**

Basic VIC Operations

- Open: Establishes link between bridge and an NIV
- Create: Sent by an IV requesting bridge to create a new virtual interface
- Delete: Sent by an IV requesting bridge to delete a virtual interface
- Enable: Sent by an IV requesting bridge to enable a virtual interface
- Disable: Sent by an IV requesting bridge to disable a virtual 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 a virtual 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 virtual interface state of a peer
- List set & list get: programs / retrieves the vif list tables in IVs

Thank You!