A Case for VEPA: Virtual Ethernet Port Aggregator

Paul Congdon
Dept. of Computer Science
University of California, Davis
Davis, California, USA
ptcongdon@ucdavis.edu

Anna Fischer
HP Labs
Hewlett-Packard Company
Bristol, UK
anna.fischer@hp.com

Prasant Mohapatra
Dept. of Computer Science
University of California, Davis
Davis, California, USA
prasant@ucdavis.edu

***Abstract* — When two virtual machines, residing on the same physical server, communicate with one another, their network traffic is not exposed to the external physical switch. For basic connectivity this is not a problem, but if the network administrator is attempting to enforce policy within the network, complications arise. Hypervisor-based virtual Ethernet switches create a change in packet flow that introduces manageability and security concerns. They also increase end-point complexity and reduce I/O performance. To obtain higher performance, the hypervisor-based software virtual switch is moving into hardware on the network interface controller (NIC). Resource and cost constraints on the NIC make it difficult for these NIC/switch hybrids to keep pace with the ever growing capabilities of the fully-featured, adjacent Ethernet switch. We propose a Virtual Ethernet Port Aggregator (VEPA) as a simple, alternative mode of operation for the embedded switch within the end-station in order to enable access to the rich set of features found in the adjacent switch. A VEPA makes it possible to simplify the management of complex network functions and limit the cost of NIC hardware while still obtaining the visibility and control of network traffic desired by network administrators. To support a VEPA the adjacent switch only requires the addition of a simple 'hairpin' mode of forwarding. We developed a software prototype of a VEPA and the adjacent 'hairpin' switch that only required a few lines of code within the Linux bridge module. The prototype demonstrates the simplicity and elegance of the solution. Performance measurements of the prototype show that VEPA is 12% more efficient than the traditional software virtual switch in environments with features enabled.**

Keywords-component; Virtualization, Ethernet, Packet Switching, Network Interface Controller, Data Center, Ethernet Bridging

I. INTRODUCTION

The complexity of data center Ethernet networking is growing due to evolving standards, new proprietary features, and the adoption of hypervisors with embedded switches. Since the original Virtual LAN standard was completed in 1998, 19 new specifications have been completed and 16 more are under development with new projects being proposed each year [1]. Beyond IEEE 802.1 specifications, there are also many IETF and vendor proprietary features that are included in a run-of-the-mill Ethernet switch.

With the advent of virtualization technology, the number of Ethernet switches has exploded and the deployment scenario has changed. The switch has moved into the end-station either in the form of a software embedded virtual switch [2] or as a low-cost hardware component on the network interface controller (NIC) [3]. A key difficulty with this approach is that traditional data center network tools lose visibility and control of network traffic as certain packet flows are no longer exposed to the traditional network infrastructure devices. Consequently, manageability and security issues arise.

The need for performance is driving a change to the switch embedded in the hypervisor. While current software-based Ethernet switches in hypervisor environments such as Xen, KVM, VMWare and Microsoft Hyper-V suffer from poor performance seen in virtual systems I/O [4, 5], direct I/O approaches [6-9] allow virtual machines to obtain efficient access to the hardware by bypassing the hypervisor. However, this direct path to hardware also bypasses the ability to implement Ethernet switching and other features in software. As a result, network interface controllers are faced with the challenge of implementing the embedded switching and inline features in a constrained and cost sensitive environment or simply not providing them at all.

The latest generation of standardized virtualization-aware PCI NICs (i.e. devices that conform to [10]) only incorporate a limited L2 switching feature set and lack other features like basic address learning, IGMP snooping, access control lists, port access control or anomaly detection. While these features are all possible to implement in software on the hypervisor virtual switch, they compound the existing performance issues and consume additional CPU cycles that were presumably purchased to support applications and not network functions. Meanwhile, all of the above features exist within the adjacent, traditional Ethernet switches. Commodity switches from Hewlett-Packard and Cisco Systems have ASIC implementations of a rich set of wire-speed networking features that are available to all ports of the device.

The current trend towards server virtualization introduces further challenges for data center networks that impede security and ease of administration. Administrators attempting to secure the data center network experience different behavior from the hypervisor-based switches. Certain packet flows are no longer exposed to the physical adjacent network devices. This results in a lack of visibility and control that introduces manageability and security concerns for network tools. Traditional network security tools lose the ability to, for example, enforce admission control, detect virus propagation, perform traffic monitoring and diagnosis, or obtain measurements. While it is theoretically possible to extend software embedded virtual switches to work with existing network security tools by implementing more advanced network policies, such an approach is costly and does not provide ideal performance as we will show in later sections.

Within the data center, the general management and configuration of traditional Ethernet switches, as well as the management of traditional servers and storage, is typically performed by departmentalized IT staff with the appropriate domain expertise. The advent of virtualization blurs responsibility across departmental lines and brings additional management complexity. For example, virtual switches are often configured by the server administrator whereas the external physical Ethernet switches are in the domain of the network administrator. The two devices, however, are both part of the same network infrastructure and must be configured consistently to assure stable and reliable operation. Network management tools depend upon consistent visibility and access to all network infrastructure devices. The sheer number of networking devices now needing to be managed expands dramatically with virtualization, thus pushing the limits of existing tools. As a consequence, vendors are developing architectures and solutions to assist in the adoption of increasingly interdependent roles for data center administrators and their toolsets [10]. One approach is to unify the management of virtual and physical machine traffic and regain widened control over data center network communication by exposing all packet flows to the external network and simplifying the capabilities within the hypervisor itself.

This paper proposes a Virtual Ethernet Port Aggregator (VEPA), a new device with a new method of forwarding network traffic between the virtualized server and the adjacent network switch. The new method of forwarding meets the low cost objectives of NICs while providing access to a rich set of networking features in the adjacent physical switch. A VEPA simplifies forwarding within the end-station by diverting all outbound traffic to the adjacent switch and thus allows for a clear division of labor between network management tasks and virtual machine management tasks. A VEPA also allows administrators to choose between high performance configurations involving local switching, but with a restricted feature set, and feature rich configurations that coordinate traffic flow with high-function external switches.

This paper builds the case for VEPA by offering the following benefits over current solutions:

1. Improves virtualized network I/O performance and eliminates the need for complex features in software hypervisor virtual switches.
2. Allows NICs to maintain low cost circuitry by leveraging advanced functions on the adjacent switch such as TCAMs, packet processors and software management subsystems.
3. Enables a consistent level of network policy enforcement by routing all network traffic through the adjacent switch with its more complete policy-enforcement capabilities.
4. Provides visibility of inter-VM traffic to network management tools designed for physical adjacent switches.
5. Reduces the amount of network configuration required by server administrators, and as a consequence, reduces the complexity for the network administrator.

This paper is organized as follows; Section II describes the details of VEPA operation. Section III describes a prototype implementation. Section IV provides an evaluation of the prototype implementation. Section V describes related work and Section VI discusses possible future work. Section VII provides a conclusion.

II. VEPA Proposal

A Virtual Ethernet Port Aggregator (VEPA) is a capability within a physical end-station that collaborates with an adjacent, external, Ethernet switch to provide frame relay services between multiple virtual machines (VMs) and the external network. A VEPA collaborates by forwarding all VM-originated frames to the adjacent switch for frame processing and by steering and replicating frames received from the adjacent switch to the appropriate VM destinations. A VEPA takes advantage of a special 'hairpin' forwarding mode on the adjacent switch to support communication between two virtual end-stations within the same physical host. Fig. 1 shows the components of a VEPA solution within a physical end-station.

The VEPA is connected to the adjacent switch by a single uplink. While it is possible to connect multiple uplinks between the VEPA and the adjacent switch, the links must be bonded or aggregated together to create a topology including only a single, direct, high-speed connection. That connection is attached to a VEPA-enabled port on the adjacent switch which supports the ability to forward frames in 'hairpin' mode (i.e. allow forwarding back out the port a frame was received). 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 virtual machines on the adjacent host to communicate with one another over the single uplink. This exception distinguishes the port attached to a VEPA uplink as a VEPA-enabled port which supports forwarding in 'hairpin' mode.

The VEPA itself may be implemented in hardware or in software on the physical end-station. Each virtual machine instantiates a virtual NIC attached to a VEPA virtual station

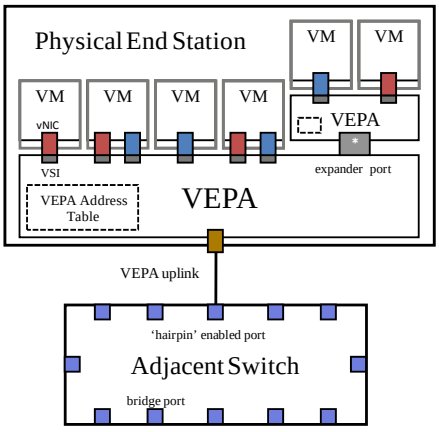

Figure 1. VEPA Components

interface (VSI). Virtual machines may have more than one virtual NIC and they may be associated with different VLANs as indicated by the shading in Fig. 1. In a hardware implementation of a VEPA, the VSIs are actual I/O interfaces within the physical machine, such as PCI virtual or physical functions as defined by the PCI-SIG [8]. They may support all of the standard features of a modern NIC including accelerators such as TCP offload, iSCSI offload, receive side scaling, interrupt coalescing, etc.

Each VEPA has an address table that holds the MAC addresses of the virtual machines attached to the VSIs. The table also holds entries for the broadcast address, specific multicast addresses and information on how to forward frames with unknown destination addresses. The table is used to determine which VSIs should receive a copy of a frame received on the uplink. Table I shows an example VEPA address table for a configuration such as in Fig. 1 where there are two VLANs and six MAC addresses associated with the six VSIs directly connecting four virtual machines.

Each entry of the VEPA address table holds a destination MAC address plus VLAN ID combination to be used as the search key. The result of an address lookup is a 'Copy To' mask that indicates which VSIs are to receive a copy of the frame received on the uplink.

The VEPA address table is populated through hypervisor configuration. While it is possible for a VEPA to dynamically learn addresses received on VSIs, it is not necessary because of the unique deployment scenario within the physical end-station. A critical distinction between a VEPA and a traditional, adjacent, Ethernet switch is that the VEPA VSIs are connected to virtual machines through a system I/O interface and, as a result, have additional information about the virtual machines themselves. The configuration of the virtual NIC by the guest operating system can be communicated to the VEPA via the hypervisor. Configuration parameters such as the MAC address, desired multicast addresses for reception, QoS settings, VLAN configuration and the intention to listen to the network promiscuously, are all available to the VEPA through a hypervisor configuration interface.

The total number of VSIs made available to virtual machines may be scaled by cascading VEPAs in a tree as shown in Fig. 1. A VSI on a root VEPA connected to a leaf VEPA higher in the topology is known as an expander port, shown on the far right. 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 know all of the MAC addresses of every virtual machine in the physical station.

TABLE I. VEPA ADDRESS TABLE

Destination MAC	VLAN ID	Copy To (vPort Mask)
A	1	1000000
B	1	0100000
C	2	0010000
D	2	0001000
E	1	0000100
F	2	0000010
Bcast	1	1100101
Bcast	2	0011011
MulticastC	1	1000100
Unk Mcast	1	1000101
Unk Mcast	2	0011011
Unk Ucast	1	0000001
Unk Ucast	2	0000001

A. *VEPA Forwarding Behavior*

The purpose of a VEPA is to multiplex the virtual station interfaces and forward their traffic to the adjacent switch for feature rich processing. As a consequence VEPAs do not forward frames directly between VSIs. If a VEPA has multiple uplinks, it will not forward frames between the uplinks. The uplinks are aggregated to form a single logical link to avoid forwarding between them. A VEPA may also be partitioned into multiple logical VEPAs, each associated with its own independent uplink.

Virtual Ethernet Bridges (VEBs) are similar to VEPAs except they allow the local forwarding between VSIs. A VEB is the general term used in this paper for a frame relay service within a physical end-station that supports local bridging between multiple VMs and optionally the external network. A VEB may be used to create an internal private virtual network between VMs or it may be used to connect VMs to the outside world. The software virtual switches within hypervisors and the more recent NIC/switch hybrids are examples of VEBs in use today.

Fig. 2 compares the forwarding behavior for both VEPA and VEB. The slashed-circles indicate forwarding paths that are not allowed. Both will forward VM traffic to the adjacent switch as needed, and neither will forward between uplinks. A VEPA does not allow local VM-to-VM communication to occur directly, and instead forces this traffic to the adjacent switch. A VEB allows local VM-to-VM communication without the assistance of the adjacent switch.

Since VEPAs do not need to learn as explained in the following sections, the egress and ingress operations can be optimized and the resources required for the address table can be minimized.

B. *VEPA Egress*

VEPA egress is defined as the set of operations required to transfer a frame from a virtual machine to the VEPA uplink. Since all frames received through VSIs are forced to the uplink, there is no need for learning MAC addresses and this operation is simple. The frame is simply moved from the VSI to the uplink for transmission as fast and as efficiently as possible. A

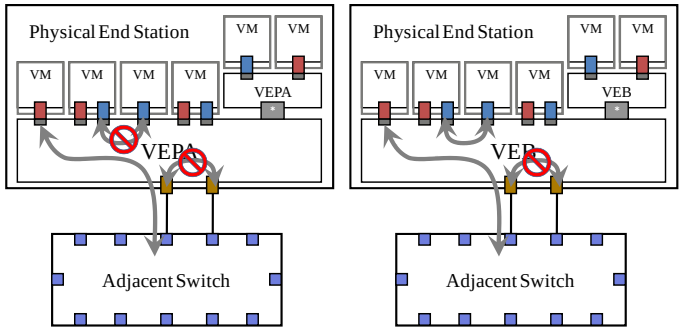

Figure 2. VEPA verse VEB forwarding

VEPA may take advantage of existing direct I/O approaches on egress since the packet is passed directly to the hardware. It is not necessary to search the VEPA address table for the VEPA egress operation. Indeed, the 'Copy To' mask from the address table in Table 1 does not include a bit for the physical uplink.

A number of optional operations may be performed on egress to enhance the VEPA solution, but these are not necessary to support the basic function. For example, the source MAC address of the virtual machine could be validated against the configuration of the VSI to prevent MAC address spoofing. Similarly the frames could be checked for proper VLAN formatting (tagged or untagged) before being admitted. It may be desirable to enforce simple QoS policies or bandwidth limits at the VSI as well.

The objective of the VEPA is to get the frames to the adjacent switch for advanced processing as quickly as possible. The adjacent switch, however, will only have the contents of the packet to work with, and in this basic VEPA mode of operation, will not actually know which VSI the frame had originated. The adjacent switch must use the source MAC address to identify the origin of the frame. An enhanced VEPA scheme that adds an explicit tag to the packet will be discussed in Section II.E.

C. VEPA Ingress

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. Similar to any standard switch, the destination MAC address is searched in the address table to locate a set of potential destination VSIs. If no entry is found, a special unknown address entry is used for either an individual address (unicast) or a group address (multicast). At the end of the address table search, the VEPA has a 'Copy To' mask that indicates which VSIs should receive a copy of the ingressed frame.

At this point, a standard switch implementation would simply use the 'Copy To' mask to deliver the frame to the appropriate VSI or set of VSIs. However, because of the 'hairpin' forwarding in the adjacent switch, a VEPA has an additional step to perform. In the case of broadcast or multicast, it is possible that one of the VSIs may have actually been the originator of the frame before the adjacent switch reflected it back. In this case, the VEPA must perform a further filtering operation to avoid delivering the frame to its originator. This step gives rise to number of considerations in the VEPA solution, but the operation is no more complex than a source address look-up required for learning.

To assure that the originator of a frame on VEPA egress does not receive a copy of the frame on VEPA ingress, the VEPA performs a source address look-up and removes any of the VSIs associated with the source address from the original 'Copy To' mask associated with the destination address. The final 'Copy To' mask can be determined as follows:

$$\text{Copy To} = (\text{Dest Copy To}) \text{ AND } !(\text{Source Copy To}) \quad (1)$$

D. The Challenge with Promiscuous VSIs

If the VEPA did not properly filter the reflected frames, a transmitting virtual machine might see its own transmitted frames, thus experiencing a different level of service when operating over a VEPA, as compared to a VEB. The VEPA must remain transparent and virtual machines must not be required to know which type of device is providing network service. In order to remain transparent, the filtering must be guaranteed to work, and the VEPA must have the MAC address of all transmitting virtual machines connected to VSIs in its address table. This poses a challenge for VSIs that are configured in promiscuous mode and potentially sourcing packets with MAC addresses not contained in the VEPA address table.

Promiscuous mode ports are used today by virtual machines that offer transparent inline services such as intrusion prevention (IPS) or content filtering. In order to remain backward compatible, a VEPA must have a way to support promiscuous ports. There are two possible solutions to this problem. The first is to require VEPAs to learn the association of source MAC addresses to VSIs as frames are received from virtual machines. This approach would require the VEPA to have sufficient address table space to hold all outstanding entries for the time it takes to reflect a frame through the adjacent switch. The second approach is to require all MAC addresses to be statically configured and validated on use – effectively prohibiting transmissions by promiscuous NICs attached to VSIs.

1) Option 1: VEPA Learning

There are several disadvantages to requiring a VEPA to learn. First, it increases the amount of memory required and thus challenges the cost and complexity of the simple VEPA forwarding behavior. Since all source addresses used by virtual machines must pass through the VSI before being reflected by the adjacent switch, the VEPA has the opportunity to learn. However, to guarantee that all recently used source addresses remain in the table once learned, the VEPA address table must be large enough to accommodate the worst case scenario. Let R be the maximum rate, in terms of packets per second, on the uplink. Let D_p be the propagation delay over the uplink and D_b be the standard worst case bridge transit delay as specified in [12]. If each address table entry is S bytes in length, then the total address table size (T_size) guaranteeing that all VSI source addresses can be held long enough to properly filter the frames reflected by the adjacent switch is given by:

$$T\_size \geq SR(2D_p + D_b) \quad (2)$$

If an address table entry is 10 bytes, a 10 Gbps uplink is a 100 meters and the standard maximum bridge transit delay is 1 second, then the address table size would need to be on the order of 148 Mbytes in order to guarantee the VEPA could always filter reflected frames properly. While this is a worst case scenario it does demonstrate that learning puts an undesirable burden on a VEPA.

2) Option 2: Static Address Table Only

If the only source addresses that are allowed to pass through a VSI are those registered statically with the VEPA, then it is possible for the VEPA to always properly filter the

frames. While this may seem like a limitation, it supports the most common configuration where virtual machines are operating as individual end-stations and not promiscuous forwarding devices. The source MAC addresses can be statically configured because the hypervisor knows the configuration of the virtual machines directly connected to the vNICs. To assure that only registered MAC addresses are used as source addresses on VSIs, the VEPA would need to perform a source address check. If a VM wishes to operate in a promiscuous mode and use unregistered source MAC addresses, the VM's VSI must be isolated from the other VEPA VSIs that are receiving reflected frames.

E. Multi-Channel Link with VEPA and VEB

Isolating VSIs can be achieved by integrating multi-channel support on the VEPA uplink. IEEE Std. 802.1ab-2005 [13] defines a bridge component called an S-VLAN component that enables the multiplexing of multiple virtual channels on a single physical link. The S-VLAN component recognizes, inserts and removes service VLAN tags (S-Tags) to enable multiple channels in the bridged network. Adding an S-VLAN component to an end-station allows VEPAs, VEBs and individual ports to operate independently and simultaneously. Each VEPA, VEB or individual port operates over its own virtual uplink instantiated by a pair of S-VLAN components; one in the adjacent switch and one on the end-station. Fig. 3 shows the inclusion of an S-VLAN component into the solution to achieve the necessary isolation to support VEPAs, VEBs and isolated promiscuous ports.

The virtual uplinks created by the end-station's S-VLAN component are effectively connected over a multi-channel uplink to virtual ports created by the S-VLAN component on the adjacent switch. 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 switch identifies the destination virtual uplink.

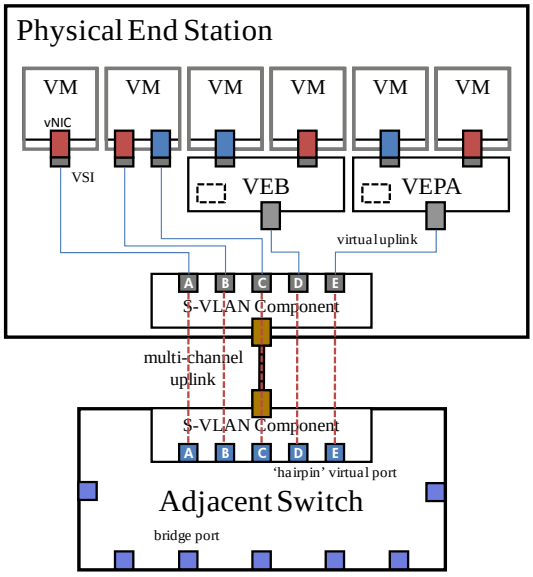

Figure 3. Multi-Channel Support

Any frames that must be broadcast or flooded to more than one virtual port are replicated by the adjacent switch 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 ports in a separate channel. By doing so, normal learning and forwarding behavior is pushed to the adjacent switch, isolating it from the simple forwarding of the VEPA. It also allows the server administrator to choose how virtual machines are connected to the network. A group of virtual machines 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 switch can be attached to a VEPA. Finally any individual virtual machine that requires an isolated promiscuous port can be attached directly to a virtual uplink.

III. Prototype Implementation Details

The Linux kernel provides built-in bridging functionality that allows configuring the operating system to act as a network switch. To prototype and test a VEPA solution, we have extended the bridge kernel module with two capabilities: it can be configured to act as a VEPA, and it can be configured to run with bridge ports being set to 'hairpin' mode. Both configurations are independent of each other and can be switched on and off independently.

The prototype implementation shows that VEPA mode only requires very minimal changes. Overall 118 lines of code have been added or modified to implement VEPA, and out of this only 37 lines of code are on the mainline data path. One significant change we have made to the forwarding path is for VEPA egress. When a frame enters any bridge port that is not the uplink port, then that frame is forwarded to the uplink port. The code accomplishing this is shown below and is located in the *br_handle_frame()* function which is the main entry point into the Linux kernel bridging code.

```
if ((br->flags & BR_VEPA_MODE) && p != br->uplink) {
   br_vepa_deliver(br->uplink, skb);
   goto out;
}
```

The function *br_vepa_deliver()* works similar to the usual bridge forwarding operation in the kernel in that it first passes frames through the netfilter subsystem so that frames can be filtered (e.g. to enforce some Ethernet-level firewalling) before they are finally transmitted on the outgoing port.

The second major extension is to enable VEPA ingress. The most significant change is on the inbound data path when a multicast or broadcast frame is received on the uplink port. The following code is an extract from the *br_flood()* function that handles the flooding of frames to multiple bridge ports while in the VEPA mode of operation.

```
if (br->flags & BR_VEPA_MODE)
     sp = br_vepa_find_src(br, eth_hdr(skb)->h_source);
prev = NULL;
list_for_each_entry_rcu(p, &br->port_list, list) {
     if (should_deliver(p, skb) && p != sp) {
          if (prev != NULL) {
      struct sk_buff *skb2;
      if ((skb2 = skb_clone(skb, GFP_ATOMIC)) == NULL) {
           br->dev->stats.tx_dropped++;
           kfree_skb(skb);
           return;
      }
      __packet_hook(prev, skb2);
          }
          prev = p;
     }
}
```

If the bridge is operating in VEPA mode then it needs to find the bridge port that originally sent this frame (e.g. assuming that frame has been reflected by the adjacent switch configured in 'hairpin' mode), and then ensures that when the frame is flooded, it is not delivered to that particular port. The VEPA does a lookup for the port using the source Ethernet address located in the frame header. The function *br_vepa_find_src()* uses the default bridge forwarding table to find the mapping of an Ethernet address to a bridge port. The function *__packet_hook()* takes care of the actual frame delivery to the bridge port(s).

The example code snippets show that the VEPA extensions mostly use existing bridge functionality. The minimal changes to implement VEPA, demonstrate that VEPA has low complexity and therefore is ideal for an implementation that can be realized in silicon (e.g. provided on NIC/switch hybrids).

The remaining code changes have been minor extensions to the existing bridge SYSFS interface, so that user-level tools can easily enable or disable VEPA capabilities. Adding the 'hairpin' mode to the bridge only required 19 lines of code to be modified with only two lines of code in the mainline data path.

Implementing a VEPA prototype as an extension to a standard Linux kernel module allows running a VEPA in a variety of application scenarios and enables easy integration with Linux-based virtualization technologies. We have published kernel patches for Linux 2.6.30 [14], which allows VEPA to be used with Linux and KVM (Kernel-based Virtual Machine) [15]. We have published patches for the Xen Kernel 2.6.18 [16]. We have furthermore posted code patches for the user-space bridging utilities [17] as well as for the user-space network configuration tools under Xen [18].

IV. EVALUATION

To evaluate the efficiency of the VEPA implementation, measurements of the throughput, delay and CPU utilization were taken while communicating between two virtual machines under the KVM virtualized environment.

The experimental setup used to evaluate performance of the VEPA proposal and prototype consisted of two physical servers based on 2.4GHz quad-core Intel Xeon X3220 processors with 4MB L2 cache. Each server was configured with six Intel e1000 Gigabit Ethernet network cards. The systems were running Fedora 10 Linux 2.6.27.24 kernels with version 74, release 10.fc10 of KVM. We used the *thrulay* [19] network performance testing tool under Linux to measure throughput and latency. CPU utilization was recorded using the Linux tool *top*.

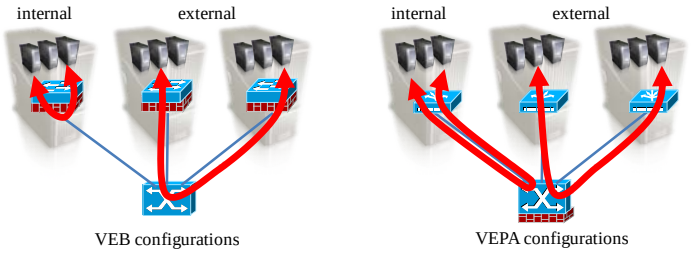

Figure 4. Basic Test Configurations

Fig. 4 shows the four basic configurations we have evaluated to demonstrate the characteristics of the VEPA proposal. In the VEB configurations, pictured on the left, virtual machine traffic is handled on each virtual switch within the physical server, while in the VEPA configurations, on the right, network processing is offloaded to the adjacent switch. We compare VEB and VEPA configurations in two deployment scenarios: internal and external. In the internal case, the two virtual machines that communicate with each other are hosted on the same physical server. In the external case the two communicating virtual machines reside on different physical servers and their traffic must traverse two ports of the adjacent switch. We obtained a total of eight different configurations by testing the performance of VEB and VEPA for both deployment scenarios; first for simple network operations (e.g. frame forwarding) and then also for heavier network processing (e.g. frame forwarding + firewalling).

Fig. 5 details the performance measurements of these eight basic configurations. The first configuration of internal VM-to-

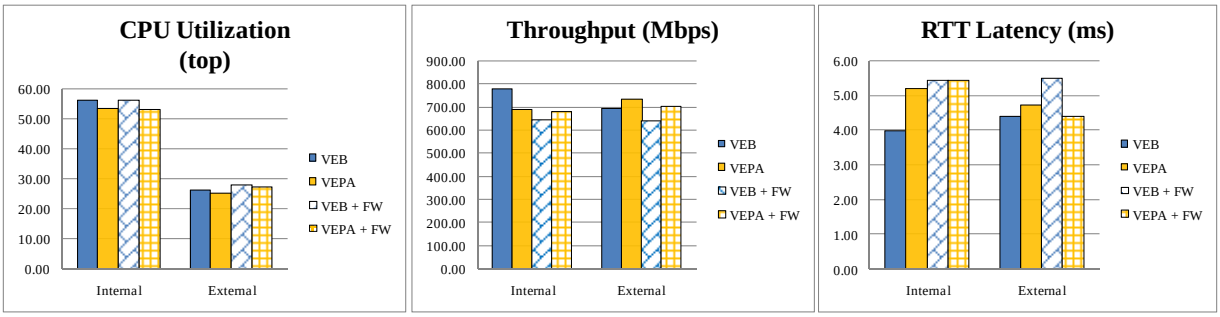

Figure 5. Performance Comparison of Basic Configurations

VM communication shows the greatest difference in performance because network traffic flows differently between a VEB and a VEPA. In these scenarios, VEB communication remains local within the physical server and VEPA communication must traverse the external link twice. Hence, additional overhead is incurred in the VEPA deployment, which most importantly, includes accessing the physical network device (e.g. interrupt processing and data copies to and from the NIC) and actually transmitting on the wire. As a result, throughput is reduced by 11.7% and latency is increased by 30.5%. However, as we enable rich inline features, like firewalling within the network path, the VEPA outperforms the VEB as it offloads these functions to the adjacent switch where rich network policies are implemented in silicon. This shows in significantly lower CPU utilization for VEPA on the server-side due to simpler frame processing operations on the machine itself. For VEB the software firewalling implementation causes increased load on the CPU resulting in an 18% reduction in throughput and a 35.6% increase in latency. As the required network processing becomes heavier, the difference in complexity and resource utilization on the server will show more clearly, and VEPA will benefit even more from leveraging services of the adjacent switch.

External VM-to-VM communication has a similar traffic flow between the VEB and the VEPA approach (e.g. both solutions now incur the previously mentioned overhead of accessing the physical network), however, VEPA achieves slightly higher throughput and uses slightly fewer CPU cycles for basic network communication. This is expected as frame handling is less costly for a VEPA than for a VEB, mainly because the VEPA does not do address learning and quickly passes frames to the external switch via the uplink port. For the configurations that additionally deploy firewall functions, the results show an even greater performance difference with throughput being 9.7% higher and latency being 20% lower for a VEPA. Here the VEPA benefits from firewalling functions being carried out in hardware on the adjacent switch, while for the VEB scenarios, firewalling is implemented in software on both virtualized servers. VEPA furthermore shows a reduced CPU load on the server for these tests, which is again, due to the VEPA only providing simple forwarding of network traffic to the adjacent switch. This means that compute resources on the server can be freed up for applications rather than being used up for network operations.

Fig. 6 shows the ratio of CPU cycles used to achieve a particular throughput as a measurement of efficiency for all the eight test scenarios. It demonstrates that VEB outperforms VEPA for internal VM-to-VM communication when only basic network functions are deployed while VEPA is more efficient than VEB for all other use cases. From these measurements we can conclude that while for internal VM-to-VM communication, the VEB approach is the preferred solution, from a performance point-of-view, the VEPA proposal improves virtualized network I/O performance significantly when communicating VMs reside on different physical servers or when rich inline network features are desired.

In a second series of performance tests, we evaluate real world scenarios where rich inline features (for example,

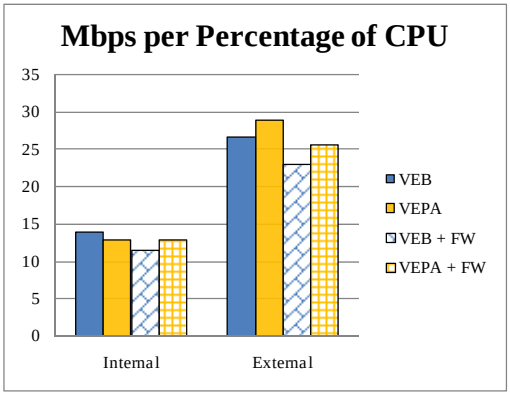

Figure 6. Basic Configuration Efficiency Comparison

firewalling) are required. For VEPA configurations, the adjacent switch implements these in silicon or a shared module, so the network traffic flow is still the same as shown in Fig. 4. For VEB configurations, these functions are implemented within a dedicated firewall VM.

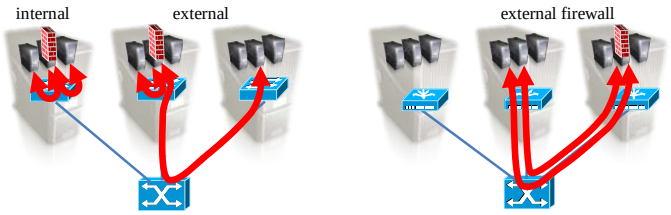

Figure 7. VEB VM Appliance Configurations

Fig. 7 shows the two additional, advanced VEB test cases that we have deployed and compared to the equivalent VEPA test cases. In the first VEB test case, as pictured on the left, the firewall VM is located on the same physical server as the communicating VM. In the second VEB test case, as shown on the right, the firewall VM is located on a different, dedicated physical server. Both these scenarios can be found frequently in typical virtualized data center deployments where network applications (e.g. firewall, network monitoring or IDS/IPS components) are often deployed as virtual machine appliances. Well-known virtual machine network appliances for these applications are provided by vendors such as Altor Networks, Vyatta, Check Point or Astaro. In the experiments, the firewall is lightly configured to permit only a small set of protocols, to verify the correctness of those protocols, and to validate the mapping of IP address to MAC address on each packet.

Fig. 8 shows the measured results from these three test scenarios. Most significantly, it shows that, for the first VEB with the firewall VM test case, throughput is very low and latency is very high compared to the other two test cases. This significant drop in performance is due to the server having to run an additional VM that carries out heavy network operations and so traffic flows into and out of that VM passing through two VEBs. This adds additional overhead of I/O virtualization and packet switching and results in increased latency and reduced throughput. Furthermore, this means that the virtualized server will have fewer resources available to serve other VMs it is hosting. The VEPA configuration, on the other hand, has low CPU utilization because the same network processing is offloaded to the adjacent switch.

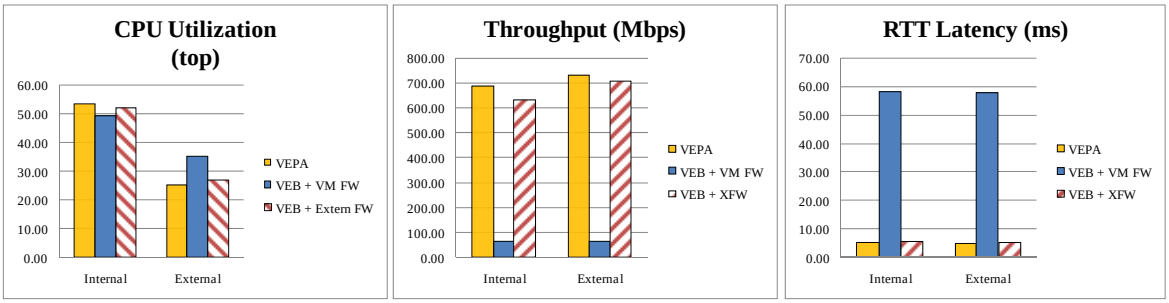

Figure 8. VEPA and VEB VM Firewall Performance Comparison

The second VEB test case runs the firewall VM on a dedicated server, so the overhead measured on the other server running the communicating VM is similar to the VEPA configuration. However, the VEPA is slightly more efficient due to less complexity in frame processing compared to the VEB. Perhaps more importantly, the VEPA uses fewer physical resources to get traffic to and from the firewall. To get traffic from multiple VMs on a separate server to the VM based firewall appliance on another server multiple physical links or creative use of VLAN isolation is required. The VEPA test case lends itself to a more optimal topology and achieves 9% higher throughput and 4.5% lower latency than the VEB approach with a dedicated firewall VM as network processing is offloaded to the attached high-function network switch.

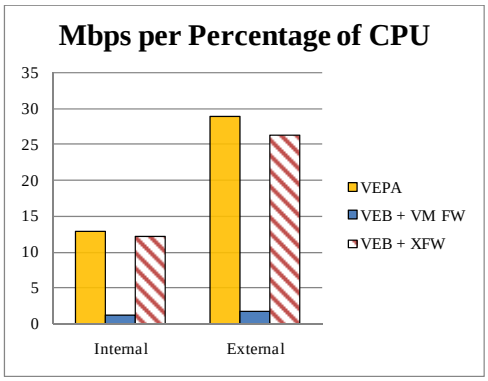

Figure 9. VEPA and VEB VM Firewall Efficiency Comparison

Fig. 9 compares efficiency measurements of the three advanced test scenarios by evaluating the achieved throughput per used CPU cycles. This final chart validates that the VEPA proposal is most efficient if rich network functions are required, while running a firewall VM imposes significant overhead on a virtualized server. The VEPA approach ensures that the server can make more compute resources available to applications as heavy-weight network processing is moved to the adjacent switch which already provides rich network functions. Additionally, the VEPA configuration uses less physical resources, such as switch ports and cables, to obtain the same treatment of network traffic.

V. Related Work

An approach most similar to VEPA is the VN-Tag proposal to the IEEE 802 working group [20]. This approach defines a new frame format that includes a tag to identify a specific virtual port within the hypervisor. Each frame traversing the link between an Interface Virtualizer (IV) and a controlling bridge contains an explicit indication of the ingress and egress virtual ports for the frame. A unique tag is created to represent a group of ports for broadcast or multicast frames and the IV is responsible for replicating the frame and delivering it to each virtual port. A VEPA is different from an IV because a VEPA contains a MAC address table and easily supports the simultaneous operation of both a virtual Ethernet bridge (VEB) and the new mode of forwarding defined by a VEPA on a single end-station. In essence, a VEPA is an additional mode of operation for an existing VEB that is a natural extension of what already exists. An IV is a new type of network device that forwards based upon a new frame tag and defines a new address space for that tag. Combining a VEB and an IV will be a burden for NIC vendors. Support for an IV on the controlling bridge also requires the creation of a large number of virtual ports and a new mechanism for identifying multicast groupings.

Other related work [21, 22] from HP Labs investigates distributing virtual networking across all endpoints within a data center. Here a software-based component resides on every server and implements network virtualization and access control for virtual machines while network switches are completely unaware of the endpoints being virtualized. The distributed software components collaborate with each other and collectively form a virtual switch [21] or virtual router [22] respectively. Network virtualization is achieved by either frame encapsulation or frame header rewriting. Both approaches differ from VEPA in that they enforce network functions in software on the endpoints and do not take advantage of rich features in the adjacent network devices. This results in significant complexity on the server side and thus lower I/O performance for virtual machines.

Virtual Distributed Ethernet (VDE) [23] is a general purpose tool for interconnecting virtual environments via virtual Ethernet switches and virtual plugs. VDE operates more along the lines of a layer-2 VPN that is connected to a virtual switch. It is a pure software environment and a LAN emulation approach that does not lend itself to the optimized hardware implementations of VEPA. One could imagine connecting virtual machines to a distributed virtual switch by using the virtual plug concept, but this topology was not discussed in the paper. Each virtual plug represents an individual virtual port on the switch and no common sharing of the port through the use of the ‘hairpin’ bridging mode is provided.

The NBAN approach described in [24] is similar to the multi-channel approach. A modified version of the standard VLAN header is used to represent a remote port where a cable modem is attached in a subscriber's home. The router at the head-end of the cable service is aggregating and routing between all of the remote ports. Since individual homes are typically not allowed to communicate directly with one another, the packet flow is similar to VEPA. However, with NBAN, the homes are also separated by layer-3 subnets. The NBAN architecture does not describe how multicast traffic is handled, but the assumption is that the head-end router replicates and routes multicast packets to each remote port. NBAN does not support the VEPA mode of forwarding at the subscriber's home and uses a redefinition of the C-VLAN tag instead of the S-VLAN tag.

VI. Future Work

Since the time of this writing, VEPA has been proposed to the IEEE 802.1 working group for standardization. A critical component to the solution that is not described here is the method to coordinate the configuration of the adjacent bridge with the VEPA and multi-channel capability. Protocols to discover, establish and enable multi-channeling, 'hairpin' mode, VEB and VEPA mode from the network side are needed to further reduce complexity on the server side. Early proposals for automating virtual bridging are being made by others [25].

VII. Conclusion

We have proposed VEPA as a new way to network virtual machines with the adjacent network. VEPA is simple and efficient. Our prototype implementation demonstrates that only minor changes are required to existing bridge architectures to support VEPA. Measurements demonstrate that VEPA is a more efficient way to apply rich networking features to the traffic generated by virtual machines. By closely collaborating with the adjacent network switch, VEPA lends itself to a network management model that empowers network administrators and relieves server administrators of complex network coordination.

VEPA can be combined with VEB through the use of physical link multi-channeling and provides administrators with additional choices on how to network virtual machines. No additional hardware resources are required to add VEPA to an existing hardware VEB so the incremental cost is minimal. Since VEPA is a natural extension of the existing VEB, it will be simple for NIC vendors to augment their NIC/switch hybrids with a VEPA mode of operation as an alternative to the inclusion of expensive and complex features.

References

[1] IEEE802, "IEEE 802.1 Home Page," 2009, http://www.ieee802.org/1/

[2] VMware, "VMware Virtual Networking Concepts," Revision: 20070718 Item: IN-018-INF-01-01, July 29, 2007.

[3] Intel, "Intel 82576 Gigabit Ethernet Controller Datasheet," 2009, http://download.intel.com/design/network/datashts/82576_Datasheet.pdf

[4] J. R. Santos, Y. Turner, G. Janakiraman, and I. Pratt, "Bridging The Gap Between Software and Hardware Techniques for I/O Virtualization," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference* Boston, Massachusetts: USENIX Association, 2008.

[5] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor," in *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*: USENIX Association, 2001.

[6] Y. Dong, Z. Yu, and G. Rose, "SR-IOV Networking in Xen: Architecture, Design and Implementation," in *USENIX Workshop on I/O Virtualization* San Diego, California, USA: USENIX Association, 2009.

[7] J. LeVasseur, R. Panayappan, E. Skoglund, C. d. Toit, L. Lynch, A. Ward, D. Rao, R. Neugebauer, and D. McAuley, "Standardized But Flexible I/O for Self-Virtualizing Devices," in *OSDI 2008: Workshop on I/O Virtualization* San Diego, CA, USA: USENIX Association, 2008.

[8] PCI-SIG, "Single Root I/O Virtualization," 2009, http://www.pcisig.com/specifications/iov/single_root

[9] L. Xia, J. Lange, and P. A. Dinda, "Towards Virtual Passthrough I/O on Commodity Devices," in *OSDI 2008: Workshop on I/O Virtualization* San Diego, CA, USA: USENIX Association, 2008.

[10] Cisco, "Unified Computing Overview," 2009.

[11] IEEE802, "IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks," New York, NY: IEEE Computer Society, 2005.

[12] IEEE802, "802.1D IEEE Standards for Local and Metropolitan Area Networks: Media Access Control Bridges," New York, NY: IEEE Computer Society, 2004.

[13] IEEE802, "802.1ad IEEE Standards for Local and Metropolitan Area Networks: Admendment 4: Provider Bridges," New York, NY: IEEE Computer Society, 2005.

[14] "[PATCH][RFC] net/bridge: add basic VEPA support," 2009, http://lkml.org/lkml/2009/6/15/415

[15] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "**kvm**: the Linux Virtual Machine Monitor," in *Linux Symposium* Ottawa, Ontario, Canada, **2007**.

[16] "[Xen-devel] [PATCH][RFC] net/bridge: Add basic VEPA support to Xen Dom0c," 2009, http://lists.xensource.com/archives/html/xen-devel/2009-06/msg01041.html

[17] "[PATCH][RFC] bridge-utils: add basic VEPA support," 2009, http://lkml.org/lkml/2009/6/15/417

[18] "[Xen-devel] [PATCH][RFC] tools: Add basic VEPA support," 2009, http://lists.xensource.com/archives/html/xen-devel/2009-06/msg01042.html

[19] S. Shalunov, "thrulay," 2007, http://thrulay.sourceforge.net/

[20] J. Pelissier, "VNTag 101," 2009, http://www.ieee802.org/1/files/public/docs2009/new-pelissier-vntag-seminar-0508.pdf

[21] S. Cabuk, C. I. Dalton, H. Ramasamy, and M. Schunter, "Towards automated provisioning of secure virtualized networks," in *Proceedings of the 14th ACM conference on Computer and communications security* Alexandria, Virginia, USA: ACM, 2007.

[22] A. Edwards, A. Fischer, and A. Lain, "Diverter: A New Approach to Networking Within Virtualized Infrastructures," in *ACM SIGCOMM Workshop: Research on Enterprise Networking 2009* Barcelona, Spain, 2009.

[23] R. Davoli, "VDE: Virtual Distributed Ethernet," in *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*: IEEE Computer Society, 2005.

[24] S. Dravida, D. Gupta, S. Nanda, K. Rege, J. Strombosky, and M. Tandon, "Broadband access over cable for next-generation services: a distributed switch architecture," *Communications Magazine, IEEE*, vol. 40, pp. 116-124, 2002.

[25] R. J. Recio and O. Cardona, "Automated Ethernet Virtual Bridging," in *First Workshop on Data Center Converged and Virtual Ethernet Switching (DC-CAVES)* Issy-les-Moulineaux, France, 2009, p. 11.