Crossbow: From Hardware Virtualized NICs to Virtualized Networks

Sunay Tripathi
sunay.tripathi@sun.com

Nicolas Droux
nicolas.droux@sun.com

Thirumalai Srinivasan
thirumalai.srinivasan@sun.com

Kais Belgaied
kais.belgaied@sun.com

Solaris Kernel Networking
Sun Microsystems, Inc.
17 Network Circle, Menlo Park, CA 94025, USA

ABSTRACT

This paper describes a new architecture for achieving network virtualization using virtual NICs (VNICs) as the building blocks. The VNICs can be associated with dedicated and independent hardware lanes that consist of dedicated NIC and kernel resources. Hardware lanes support dynamic polling, which enables the fair sharing of bandwidth with no performance penalty. VNICs ensure full separation of traffic for virtual machines within the host. A collection of VNICs on one or more physical machines can be connected to create a Virtual Wire by assigning them a common attribute such as a VLAN tag.

Categories and Subject Descriptors

D.4.4 [**Operating Systems**]: Network communication; C.2.4 [**Computer-Communication Networks**]: Network operating systems

General Terms

Design, Performance, Security, Experimentation

Keywords

Virtualization, Networking, Performance, Hypervisor, VMs, Zones, VLAN, Classification, Crossbow, vWire, VNICs

1. INTRODUCTION

Virtualization has seen tremendous growth in the last few years. Hypervisor-based virtualization [2] and operating system-based virtualization [18] [19] are two popular virtualization technologies. More recently, I/O virtualization (IOV), an exclusively hardware-based device and bus virtualization technology [17], has also emerged.

Now that server virtualization has become mainstream, the focus has shifted to network virtualization, where the deployment model requires no interference between the network traffic of different virtual machines. An ideal scenario would be to assign a physical NIC to each virtual machine. However, the number of I/O slots available on a machine is fairly limited, and the cost per virtual machine would increase due to the additional required power consumption, physical connectivity, and administration overhead. Thus, there is a need to securely share the NIC among the virtual machines in a fair or policy-based manner.

Crossbow is the code name for the new OpenSolaris networking stack that supports virtualization of a physical NIC into multiple VNICs. It aggressively uses the NIC hardware features for performance, security isolation, and virtualization. A VNIC is assigned a dedicated *hardware lane*, which consists of NIC resources such as receive and transmit rings, dedicated software resources, and CPUs. These dedicated resources establish an independent data path, which is free of contention. If the hardware resources are exhausted, or if the NIC does not support virtualization, then the stack falls back to software based NIC virtualization, albeit at an extra performance cost.

In this paper, we first take a look at the problems with existing virtualization solutions. We then survey the recent NIC hardware advancements that enable us to build the Crossbow architecture. In Section 4, we describe the Crossbow architecture itself, and in Section 5, we show how it is used to build a *Virtual Wire*, a fully virtualized network connecting virtual machines spread across one or more physical machines. Finally, we look at other work happening in this area and describe our future direction.

2. ISSUES IN EXISTING ARCHITECTURES

First consider some of the key requirements for network virtualization and how existing hypervisor based and fully hardware based solutions (IOV) attempt to meet those requirements.

2.1 Fair Sharing

The ability to support fair sharing of the underlying physical NIC among its constituent virtual NICs is a key network virtualization requirement. This is an issue for both hypervisor-based virtualization and IOV virtualization. In both cases, a virtual machine can monopolize usage of the underlying physical NIC resources and bandwidth, if it is capable of driving the physical link at line rate.

Schedulers have focused mostly on sharing the CPU resources, and network packet scheduling has been mostly left to the NIC drivers, which are unaware of higher level services or virtual machines. In [14] the authors discuss the prob-

VISA'09,, August 17, 2009, Barcelona, Spain.
ACM 978-1-60558-595-6/09/08.

lem of the scheduler not achieving the same level of fairness with I/O-intensive workloads as they do with CPU-intensive workloads.

2.2 Security

In the IOV model, a virtual machine is granted direct and uncontrolled access to partitions of resources on the NIC, thus maximizing the performance. However, this model has security implications. In particular, firewall rules are not handled by the IOV Virtual Functions (VFs), which opens the door for a virtual machine to spoof its MAC or IP addresses, generate bridge protocol data units (PDUs), or fake routing messages and bring down the entire layer 2 or layer 3 network.

This issue is especially critical in cohosted environments where multiple tenants share the same hardware.

2.3 Performance

Performance is an issue for the hypervisor-based model [12] [15]. On the receive side, considerable effort is spent on bringing the packet into the system and performing software classification before the destination virtual machine can be identified. Then, the traffic needs to be passed to the virtual machine through the hypervisor, which is also expensive.

In [15] the authors report substantial performance impact and higher CPU utilization. In [24] [11] the authors present specific performance optimizations such as using NIC Hardware checksum and TCP Large Segment Offload, and report significant gains. However, they point out that the performance impact is still substantial, especially on the receive side.

2.4 Virtual Machine Migration

The hypervisor based solutions support migration of virtual machines from one host to another, but IOV-based solutions don't easily lend themselves to virtual machine migration, since the virtual machine has a state associated with the bare metal.

3. HARDWARE ADVANCEMENTS

Most modern NICs [7] [20] [13] support a combination of hardware features, which are described in this section.

3.1 Multiple Receive and Transmit Rings and Groups

One of the main features of NICs is the support of multiple receive and transmit rings. Each hardware ring or queue has its own descriptors, and can be assigned its own resources on the bus (DMA channels, MSI-X interrupts [17]) and on the system (driver buffers, interrupted CPU). Multiple CPUs can therefore cooperate in receiving and sending packets, which is particularly needed when a single CPU is too slow. Rings can be bundled in statically or dynamically formed groups.

On the receive side, one or more MAC addresses, VLAN tags, or both, can be associated with a ring group. A steering logic on the NIC can deposit incoming packets to any ring of the group that matches the packets' address or VLAN. A load balancer (also known as Receive-Side Scaling (RSS) engine) or a higher level classification engine determines the actual recipient ring based on a matching hash value or specific L3/L4 classification rules. Packets that do not match any programmed MAC address (for example broadcasts) or classification rule are delivered to a default ring on the NIC.

On the transmit side, multiple rings enable the sending of multiple packet flows through the same device in parallel. Similar to receive rings, transmit rings can be bundled together. A transmit rings group is a set of transmit rings with the same capabilities (hardware checksum, LSO, and so on) and a transmit load balancer. As of the time of writing this paper, only few vendors have announced future hardware support of transmit ring grouping.

An advanced virtualization feature is the capability to create a partition on the NIC with its own receive and transmit ring groups, as implemented by Intel's Virtual Machine Device Queues (VMDq) [8].

3.2 SR I/O Virtualization

The PCI-SIG consortium developed the Single Root I/O Virtualization and Sharing (SR-IOV) [16] specification primarily to address the issue of platform resource overhead from the hypervisor trap imposed on all I/O operations between virtual machines and devices, while preserving the capability to share I/O devices among multiple virtual machines. A device maps to a single physical function (PF) on the bus, and can be partitioned into multiple virtual functions (VFs). Independent interrupts, rings, and address translation services allow virtual domains to control their dedicated VF directly. Some NICs also offer on-board switching of traffic between VFs.

4. CROSSBOW ARCHITECTURE

One of the key requirements of network virtualization is to ensure that virtual machines are insulated from each other's traffic.

As mentioned in Section 1, true isolation of a virtual machine and its network traffic can be achieved by dedicating a physical NIC and its associated network cable and port on the switch to the virtual machine itself. If the MAC layer can have dedicated resources for each physical NIC (without any shared locks, queues, and CPUs) and the switch ensures fairness on a per port basis, the traffic for one virtual machine will not interfere with the traffic of another virtual machine.

In the case where the physical resources, and in particular the physical NIC, needs to be shared among multiple virtual machines, the next best option is to virtualize the NIC hardware and the layer 2 stack such that the sharing is fair and without any interference. The Crossbow architecture in the OpenSolaris OS does exactly that by virtualizing the MAC layer and taking advantage of NIC hardware capabilities to ensure traffic separation between multiple virtual machines.

4.1 Virtualization lanes

A key tenet of the Crossbow design is the concept of *virtualization lanes*. A lane consists of dedicated hardware and software resources that can be assigned to a particular type of traffic. Specifically, it consists of the following:

- NIC resources such as receive and transmit rings, interrupts, and MAC address slots.
- Driver resources such as DMA bindings.
- MAC layer resources such as data structures, locks, kernel queues and execution threads to process the

packets, and CPU bindings of kernel threads and interrupts. Crossbow carefully instantiates locks and counters per lane rather than sharing them across lanes to avoid any contention.

A virtualization lane can be one of two types, hardware-based or software-based.

Hardware-based virtualization lanes – In this case, the NIC must support partitioning of its hardware resources. At a minimum, a hardware-based lane needs to have a dedicated receive ring. Other hardware resources such as a transmit ring can be exclusive or shared among lanes. One or more lanes can in turn be assigned to a virtual machine. Incoming packets are dispatched based on fair scheduling across all hardware-based lanes. Alternatively incoming packets can be dispatched based on administrative policies, such as a bandwidth limit.

As mentioned in Section 2, the bulk of the receive processing consists of bringing the packets into the system and inspecting the packet headers. By assigning each hardware lane its own receive ring, the incoming packet arrival rate for a given lane has no impact on other lanes. On the transmit side, the hypervisor or kernel knows where the packets originate from, and hence can support fair sharing in software. Therefore, a hardware lane does not require its own dedicated transmit hardware rings.

Software-based virtualization lane – A set of default receive and transmit rings can be shared by multiple VNICs to handle the situation when a NIC runs out of hardware resources. In such situations, software-based receive and transmit rings, also called *softrings*, are used and the virtualization lanes are referred to as *software-based virtualization lanes*. The system can always operate in mixed mode where all but one hardware receive rings are assigned to hardware lanes, and the last receive ring is software classified to build software lanes. Although there is no limit to the number of software lanes that can be built, software-based lanes sharing hardware do not have some of the fairness and isolation attributes of hardware lanes. Legacy NICs without hardware partitioning capability appear as NICs with single receive and transmit rings and allow only software-based lanes.

4.2 Virtual NIC

To complete the network virtualization abstraction, Crossbow enables the creation of Virtual NICs (VNICs) that are associated with their own hardware or software lanes.

The VNIC appears like any other NIC and is administered exactly like a physical NIC. In fact, the physical NIC seen by the higher layers of the network stack is nothing but a VNIC built over the physical NIC, albeit with the well known name (such as 'nxge0') of the physical NIC itself. In Crossbow architecture, the NICs, VNICs, and link aggregations are all treated similarly and are termed as a data link.

A lane is assigned to a VNIC and the classifier is programmed based on the VNIC's MAC address and optionally the VLAN tag, such that packets destined for the virtual machine end up in a receive ring dedicated to that virtual machine and can be scheduled appropriately. Receive load balancing can be used to distribute packets among multiple receive rings for better performance, see Section 3.

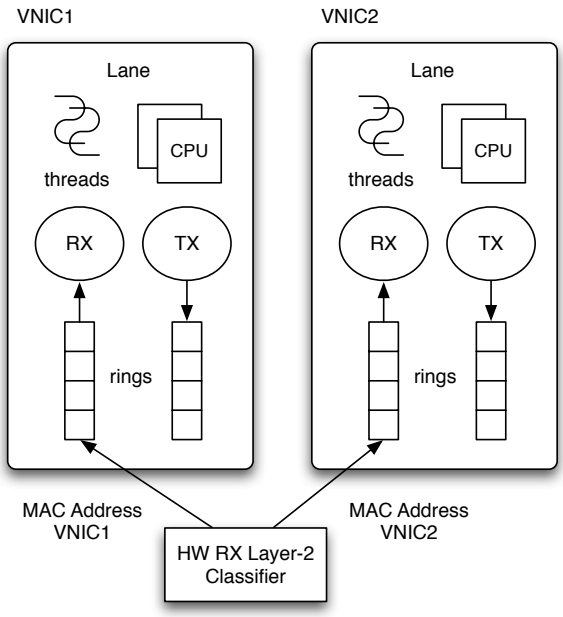

Figure 1: Virtualization lanes with dedicated hardware and kernel resources

When VNICs are created with a VLAN tag, Crossbow can use the GVRP [6] or MVRP [5] protocols to automatically register the VLAN tag with the physical switches as well.

4.3 Dynamic Polling and Packet Scheduling

The Crossbow architecture introduces the concept of *dynamic polling*, where instead of a fixed per packet interrupt, the virtualization lane is switched between interrupt and polling modes. Under low load, where the packet arrival rate is less then the packet processing time, the lane stays in interrupt mode and the receive ring is programmed to issue an interrupt as soon as any packet is received without applying any local optimizations. As soon as a backlog builds up, the lane is dynamically switched to polling mode. In this mode, a kernel thread goes down to the receive ring in the NIC hardware to fetch all packets in the form of a packet chain.

Each virtualization lane operates independently from other lanes and does not require any locks for synchronization. In addition, the NIC receive ring serves as the only queue with no additional queuing within the stack. Typically, three kernel threads are used to process packets within a lane:

- *Poll thread* that goes to the NIC hardware to retrieve packets in the form of a packet chain. Under low to moderate load, the poll thread can run to completion without passing the packet chain to the worker thread, thus reducing the overall system context switches. If a backlog builds up at higher layers, the receive ring stays in polling mode, and the worker thread is used to send the packet chain up the stack. Because the worker thread often runs on a different CPU than the poll thread, the throughput can be increased significantly,

	intr/sec	csw/sec	smtx/sec	idle
pre-Crossbow based	10818	4588	1797	12
Crossbow based	2823	875	261	27
percent change	-75	-85	-85	+125

Table 1: Measured reduction in interrupt/sec, context switch/sec, lock contention/sec, and increase in idle time due to dynamic polling

albeit with an context switch on a per packet chain basis.

- *Worker thread* that does the receive protocol processing (IP and above) or delivers the packets to the virtual machine through a back-end driver. The worker thread also does any transmit work that is a side effect of the receive processing, such as forwarding an IP datagram or doing ACK driven transmit in the case of TCP.
- *Transmit thread* that gets activated if packets are being sent out as a result of transmit side flow control relief, or after transmit descriptor reclaim. As long as there are transmit descriptors available and protocol permits, the application or virtual machine threads are allowed to transmit packets without any context switches or additional queuing due to flow control.

Because, under load, large number of packets are delivered in chains using the polling mode, the cost of going through the stack is amortized across the entire chain. Also, in polling mode, overheads due to context switching, mutex contention, and thread pinning are eliminated because interrupts are turned off. The higher the packet per second rate, the better is the measured throughput and lower is the CPU utilization. Moreover, because the hardware already handles the classification and guarantees that the packets are indeed unicast packets meant for the MAC address of the VNIC, the entire data link processing can be bypassed, further reducing the packet-parsing overheads. Table 1 compares the rate of interrupts per second, context switches, and mutex contention without and with dynamic polling. As shown by the table, dynamic polling and hardware lanes offer very significant reduction in context switches and lock contention, and decrease CPU utilization. The workload was a standard web based workload on a low-end server over a 1 Gigabit Ethernet link with 12 clients connected via an Extreme Networks Gigabit Ethernet switch. The average bandwidth utilization was 500-550 Mb/s, with most of the transmitted packets being larger size (800 to 1500 bytes), and most of the received packets being smaller sized or just ACKs.

To obtain better throughput for high bandwidth data links that have only one or small number of hardware based virtualization lanes, a software-based fanout can be employed on a per lane basis using softrings. A *softring* is a packet serialization queue with an associated worker thread. The poll thread just queues packets in one of the softrings based on a pre-configured hash, and the softring worker thread does the protocol processing and any transmission. All the threads are assigned to independent hardware execution units (cores, hardware threads, CPUs) for better pipeline performance whereby the poll thread just brings packets into the system, and the softring worker threads do the protocol processing.

For optimal performance without software-based fanout, high bandwidth data links (10 Gb/s or higher) require the use of multiple hardware-based virtualization lanes, each assigned two to three dedicated CPUs. The poll thread and interrupt share a CPU since they are mutually exclusive from each other. The worker and transmit threads need a dedicated CPU, although NIC hardware that support large segment offload (LSO) do not need a dedicated CPU for the transmit thread.

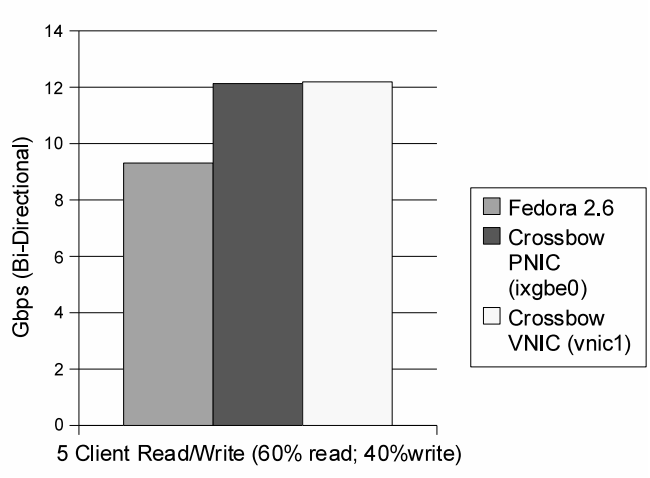

Figure 2: Comparing VNIC and physical NIC performance

In the case of software-based fanout, additional CPUs are needed for each softring to allow for concurrent processing. There are overheads due to software-based fanout, such as walking the headers to compute the hash, context switches, and so on, but more packets can be processed at the cost of higher CPU utilization. Threads assigned to a lane can share CPUs if there are administratively configured policies in place for the data link, or if the system is short on CPUs.

4.4 Virtualization Performance

VNICs are closely integrated with the MAC layer architecture to provide virtualization at no extra cost when they are assigned one or more hardware-based virtualization lanes. Figure 2 shows the bi-directional aggregate throughput of a Crossbow physical NIC under stress as compared to Fedora 2.6. The figure also shows the throughput when a VNIC is used exclusively in the place of a physical NIC. Note that no performance difference exists between the physical NIC and the VNIC.

The test setup consisted of six identical machines where one machine acted as a system under test (SUT) and the other five machines acted as clients. Each machine was a dual socket, quad core Intel machine with each core operating at 2.8 GHz. All six machines had an Intel 10 Gigabit Ethernet NIC and were connected through a dedicated Foundry 10 Gigabit Ethernet switch. Each client was sending or receiving 10 TCP streams to the SUT, where the total ratio of transmit to SUT and receive from SUT was 2:3. The wire MTU was 1500 bytes, and the application write buffer size was 8 Kbytes. We used an application called uperf [1], which is similar to other micro benchmarks but enables us to deal with a multi-client case more effectively, thus better simulating a real-world performance. Each test was run for 10 minutes.

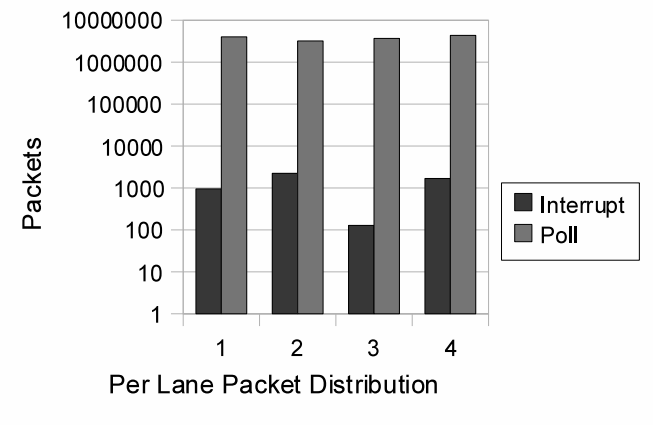

Figure 3: Packets received through interrupts vs poll path

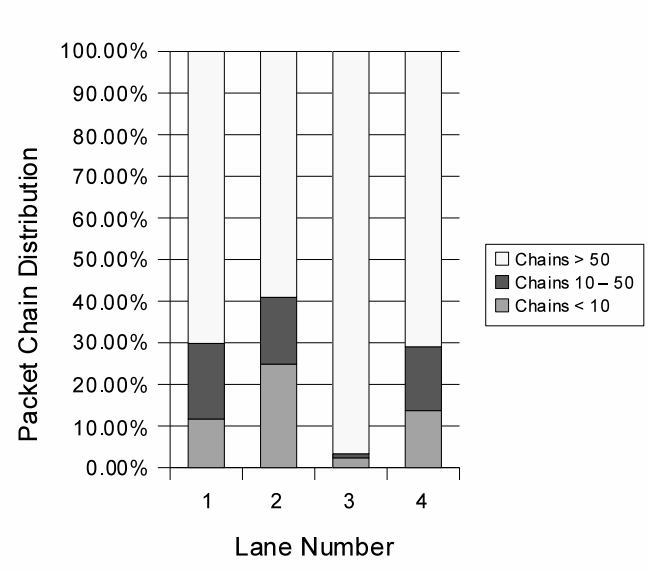

Figure 4: Chain lengths when packets are received through polling mode

The Intel 10 Gigabit Ethernet NIC [7] has 8 receive, 8 transmit rings, and support large segment offload as well. Fedora 2.6 used all 8 receive rings and 1 transmit ring. Crossbow used only 4 receive rings with one hardware based lane for each ring, for a total of 4 hardware-based virtualization lanes. As mentioned before, by default, Crossbow prefers to use at least two dedicated CPUs per lane (one CPU for poll and interrupt thread, and the other for protocol processing) and since the system under test has only 8 CPUs, it enables only 4 lanes. All 4 lanes shared 1 transmit ring. Because TCP large segment offload was enabled on both Fedora and Crossbow, a single transmit ring was sufficient and Crossbow didn't need a dedicated CPU for transmit processing. The Crossbow numbers over multiple runs gave very consistent results, and the load across all 8 CPUs was very evenly distributed. In the case of Fedora 2.6, multiple runs gave very inconsistent results with some as low as 5.6 Gb/s, and the system consistently pegged 3 to 4 CPUs to 100 percent load. Note that the unidirectional TCP throughput of both Crossbow and Fedora 2.6 were fairly similar and very close to the line rate of a single 10 Gigabit Ethernet NIC. The goal of the experiment was not to compare performance between OpenSolaris and Fedora, but rather to showcase the concept of lanes and demonstrate that Crossbow VNICs don't impose any performance penalties.

Figure 3 shows the ratio of total packets that were received through the interrupt path as compared to the polling path. In general, the interrupt path is only used when a lane has no packets to process. In this case, the per packet interrupt is turned on for better latency. As soon as the backlog builds up, the packets can't be processed inline and need to be queued. The lane then switches to polling mode. A kernel thread then goes to the NIC hardware to pick up the packets in the form of a chain instead of relying on per packet delivery through the interrupt path. Figure 4 shows the distribution of packet chains length received during polling mode on a per hardware lane basis.

4.5 Thread to CPU binding and NUMA interactions

The number of lanes required depends on the packet per second rate rather than the throughput. On low bandwidth data links where link speed is a few Gigabit per second or lower due to physical limitations (100 Mb/s or 1 Gb/s NICs) or is administratively limited, one hardware-based virtualization lane is adequate. For higher bandwidth data links (10 Gb/s or more), 4 to 8 virtualization lanes are required for processing small packets at line rate.

Although, the Crossbow architecture allows linear scaling using multiple hardware based virtualization lanes, it introduces new challenges with respect to thread binding. With large packet sizes, a 10 Gb/s data link requires only 1 or 2 hardware lanes to sustain line rate. With small packets sizes, the same data link can require 8 lanes or more to achieve line rate. Since each lanes needs two to three CPUs for optimal processing, the total number of CPUs can be 16 to 24 CPUs, and with multiple 10 Gb/s NICs configured on the system, a typical x64 system can quickly run out of CPUs.

On platforms based on Non Uniform Memory Access (NUMA) architectures, CPUs for a given physical NIC hardware are not equal. This makes the thread to CPU binding a major challenge. In such cases, the architecture needs to take the NUMA topology and location of the physical NIC into account for thread placement.

The Crossbow administrative interfaces allow a system administrator to specify CPUs for a data link to allow administrative control over the thread binding. In the case of virtualization, where CPUs are assigned to virtual machines or zones, the Crossbow architecture can bind the lane threads to the same CPUs. A more dynamic thread placement approach is needed to optimize the number of lanes and threads per lane based on workload, packet per second rate, NUMA topologies, and any administrative policies.

4.6 Bandwidth Partitioning

Apart from the fair sharing of bandwidth, Crossbow enables the administrative configuration of link speed for each VNIC. The link speed is implemented by regulating the periodic intake of incoming packets per virtualization lane. Every period, the network stack allows only as many packets

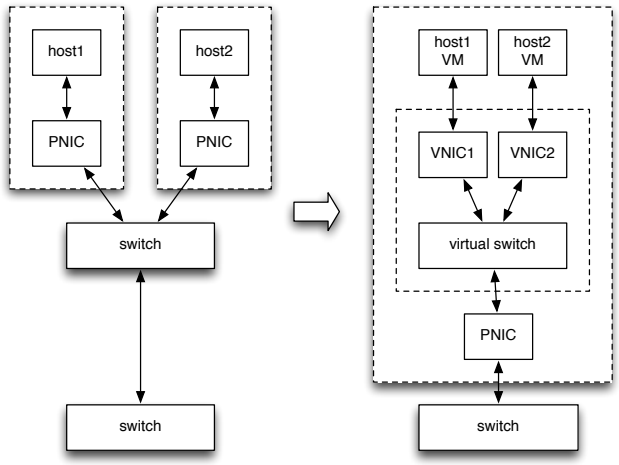

Figure 5: Mapping between physical and virtual switches

to come into the system as permitted by the configured link speed. As soon as the lane consumes enough packets for a given period according to the configured link speed, the lane abstains from picking up more packets until the next period.

The transmit bandwidth is much easier to control because packets originate from the system. In that case, the network stack can either flow control the application generating the packets or drop the excess of packets.

The same bandwidth control mechanisms are used to implement QoS for traffic flows, as described in [21].

4.7 Virtual Switching

When multiple VNICs share the same underlying physical NIC or link aggregation, the MAC layer provides a data path between these VNICs. This data path enables the VNICs to exchange packets as if they were connected to a *virtual switch*, without leaving the machine for better performance, and also because a physical Ethernet switch port would not loop back a packet to its originating port. Figure 5 shows the mapping between physical and virtualized network components.

Note that a NIC can also be plumbed by IP while it is virtualized with VNICs. In this case, virtual switching also enables the IP stack to communicate with the VNICs defined on top of the same NIC. As such, IP and VNICs are referred to more generically as *MAC clients* of the underlying NIC. The rest of this discussion refers to VNICs, but the same concepts apply to other MAC clients in general.

Virtual switches are never directly accessed by, or visible to, the user or system administrator. Instead, they are implicitly created when the first VNIC is defined on top of a NIC. Virtual switches provide the same semantics that are expected from physical switches, as follows:

- VNICs created on top of the same underlying NIC can send Ethernet packets to each other using their MAC addresses.
- Virtual switches distribute multicast and broadcast packets between VNICs. They maintain multicast memberships to find the appropriate destinations.
- Virtual switches are VLAN aware. They maintain per VLAN broadcast domains on top of the same physical NIC, which allows separation between VLANs in a virtual environment.

In addition, virtual switches enable not only the switching of traffic between VNICs, but also the demultiplexing of inbound traffic coming from the wire that was not classified in hardware, as well as the sending of traffic for non-local destinations.

4.8 Etherstubs

As described in the previous section, the MAC layer provides the virtual switching capabilities that allow VNICs created on top of an underlying data link such as a physical NIC or a link aggregation. The local data paths implemented by the MAC layer provide the same semantics as a layer 2 switch to which the VNICs are virtually connected.

In some cases, it is preferable to create virtual networks on the same machine without the use of a physical NIC. Some examples are described in Section 5. Because a virtual switch is implicitly created when a VNIC is created on top of a data link, one solution is to have the capability to create pseudo ethernet NICs, and to define VNICs on top of these pseudo NICs. Crossbow provides such pseudo NICs in the form of *etherstubs*. An etherstub is a pseudo ethernet data link that can be used instead of a physical NIC or link aggregation during the creation of a VNIC.

4.9 Virtual NIC and Virtual Machine Migration

One goal of the Crossbow architecture is to maximize the use of the hardware features while still allowing for fallback to software, in the absence of hardware features. In [9] the authors point out that Crossbow's hardware-based virtualization can inhibit portability and virtual machine migration. However, our architecture is, in fact, a hybrid model of hardware and software virtualization where the virtual machine itself has no knowledge of the NIC hardware. The architecture allows a hardware VNIC to be moved to a NIC that doesn't support virtualization, thus allowing a virtual machine to be easily migrated.

4.10 Monitoring VNICs and Virtual Switches

The OpenSolaris Operating System provides the capability through snoop(1M), DLPI, and other APIs to monitor the traffic sent and received by a data link such as a physical NIC. When a data link with VNICs is monitored, the traffic observed includes packets sent and received through the wire, as well as all traffic exchanged between the VNICs created on top of that data link, for all VLAN IDs.

When a VNIC is monitored, the observed traffic includes the traffic that would be normally seen by a host connected to a physical switch, that is all multicast and broadcast traffic, and the unicast traffic for the unicast address and VLAN ID associated with the VNIC.

5. VIRTUAL NETWORKS AND VIRTUAL WIRE

Section 4.7 explained how the MAC layer creates virtual switches for each data link on top of which virtual NICs have been created. VNICs can be created on top of physical NICs and link aggregations, but also on top of etherstubs.

Etherstubs enable the VNICs to communicate through virtual switches that are completely independent from hardware. Because many etherstubs can be created on a single host, VNICs and virtual switches can be easily combined to build *virtual networks in a box*, also known as *Virtual Wire*.

This section presents some use cases for Virtual Wire, describes how VLANs can be combined with Virtual Wire to build isolated virtual networks sharing the same physical infrastructure, and concludes with two examples to illustrate these concepts.

5.1 Use Cases

The Virtual Wire functionality of Crossbow can be used in many scenarios, some of which are described below.

P2V for Distributed Applications – As an extension of the *P2V* (*Physical to Virtual*) capability, virtual wire also enables the network connecting the hosts being virtualized to be simulated in the virtual environment. This enables distributed applications to be virtualized without changes to the applications, or to the network topology connecting the hosts being virtualized.

Simulation – The design of a distributed application environment and network topology is typically time consuming, and requires cycles of design, implementation, and testing before going into production. Virtual Wire enables these network topologies to be virtualized, which allows for the development and verification cycles to be performed in a virtual environment, on a developers laptop at the local coffee shop, or on a single machine in a data center. Once a network topology is tested, it can be instantiated on a physical network.

Security – Virtual Wire can be used to build private networks running on a single machine that are isolated from the rest of the network by a software firewall running in a virtual machine or Solaris zone. Firewall capabilities, network address translation (NAT), and so on, can be provided by existing Solaris services or by virtual machines connected to the multiple virtual networks.

Education and Research – The network developers and researchers working with protocols (such as high speed TCP) can use OpenSolaris to write their implementations and then them with other production implementations. They can debug and fine-tune their protocol quite a bit before sending even a single packet on the real network.

5.2 Virtual Wires Separation

Virtual Wire also enables virtual networks to be extended to span multiple physical machines connected by a physical network. Several such networks can co-exist on a network, and their traffic can be separated using techniques such as VLAN tagging. A VLAN tag can be assigned to a Virtual Wire, and each Virtual Wire instantiates a virtual network.

As seen in Section 4.2, a VNIC can be assigned VLAN tag. When a Virtual Wire is constructed, every created VNIC is assigned the VLAN tag of the Virtual Wire it is connected to. Multiple VNICs with the same VLAN tag can be connected to the same virtual switch, which allows VNICs on the same VLAN to exchange traffic. The resulting Virtual Wire topologies can consist of multiple VMs or zones per physical hosts, as well as VMs or zones residing on different physical hosts.

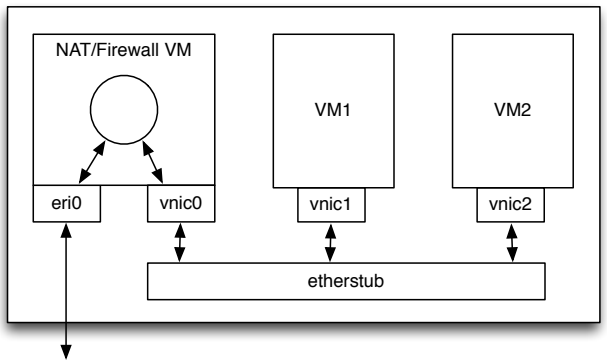

Figure 6: Virtual wire example: building a network in a box

The use of VLANs requires packets to be tagged with a VLAN header as they flow through the virtual and physical networks. With Virtual Wire, VLAN tagging and stripping is done by the VNICs and MAC layer of the host operating system transparently to the VMs and zones connected to the Virtual Wire. From their point of view, the VMs and zones send and received untagged Ethernet packets, and appear to be connected to their private virtual network. The separation between Virtual Wires, tagging and stripping of packets, as well as virtualization are implemented by the host operating systems of the physical nodes.

This architecture also allows the VLAN separation, tagging, and stripping to move from the physical switches into the host. This allows the provisioning of isolated virtual networks, even when they span multiple physical machines, to be done without reconfiguring the physical switches, and facilitates the migration of virtual machines between multiple physical hosts.

5.3 Examples

Figure 6 shows a Virtual Wire example where an etherstub is used to create a virtual switch connecting three VNICs, $vnic_0$, $vnic_1$, and $vnic_2$. One virtual machine (VM) of the host is connected to the physical network through a physical NIC, here eri_0, and at the same time to the private virtual network through $vnic_0$. That VM implements NAT and a firewall for the other virtual machines, and in the case of Xen could be dom0 itself. The other VMs VM_1 and VM_2 are connected to that private network through their VNICs, $vnic_0$ and $vnic_1$, respectively.

Figure 7 shows two networks, Virtual Network A and Virtual Network B, sharing the same common physical switch and spanning multiple physical hosts. On each host, multiple virtual machines can be connected to each one of these networks through VNICs. Crossbow enables the VNICs to be assigned to a specific VLAN, which ensures that a virtual machine will only see the traffic for its network.

6. RELATED WORK

Network Virtualization has been around in some form or the other for quite some time. One of the early virtualiza-

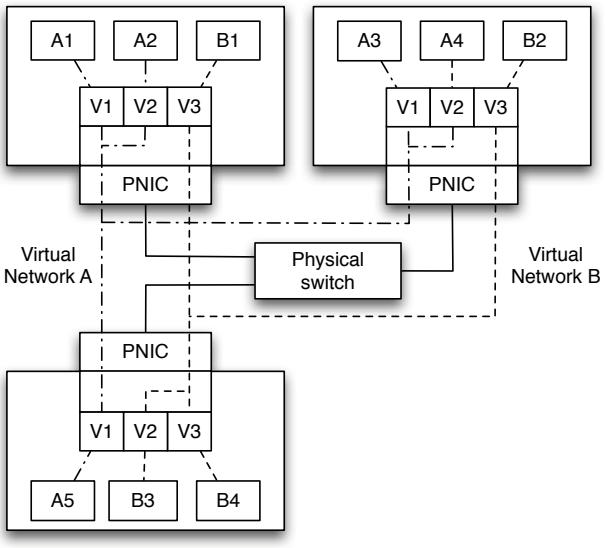

Figure 7: Virtual wire example: virtual networks spanning multiple hosts separated by VLANs

tion technology was FreeBSD Jail [4] which is an Operating System based virtualization technology that provides strong partitioning and isolation for applications. Clonable network stacks [23] provides the network interface virtualization on FreeBSD. This is a simple software-based scheme that uses bridging to connect multiple network stacks and the underlying physical link. Solaris zones [18], an application container technology is similar, but provides deeper and more complete system integration and uses Crossbow architecture for network virtualization which focuses on performance and use of hardware capabilities.

The last decade saw a significant advent of hypervisor based system virtualization in mainstream deployments and on commodity hardware. In recent years, the trend is shifting from desktop virtualization to server virtualization with more intensive networking workloads. Some of the most popular virtualization technologies addressing server and network intensive virtualization are Citrix System Xen, VMware ESX, and Microsoft Hyper-V. More recently, Cisco Systems announced a virtualization offering based on VN-Link technology. In this section, we will take a closer look at some of these virtualization technologies.

Citrix System Xen [2] based Virtual Machine Monitor (VMM) has been implemented natively in the Linux kernel, and the code is available in open source. In the Xen based VMM, the physical NIC is owned by the hypervisor or a special privileged domain called driver domain. Virtual machines access the network through a virtualized network interface which has a front-end driver in the guest domain, and a back-end driver in the privileged domain. Within the privileged domain, an Ethernet bridge is used to demultiplex incoming packets to destination virtual machines. The bridge forces the NIC to go in promiscuous mode and has significant overhead for network intensive workloads, as pointed out by Menon et. al. [12]. The same authors have also suggested optimizations [11] which primarily focus on optimizing the data transfer between the front-end and back-end drivers. The cost of demultiplexing incoming packets and allowing the available bandwidth to be shared in a fair manner under diverse network workloads still needs to be addressed.

VMware ESX and Microsoft Hyper-V based VMM are not open source and are proprietary implementations. As such their specific implementations are hard to quantify, but some earlier work done by Sugerman et. al. [22] suggests that ESX VMM also needs to address issues related to demultiplexing inbound packet processing and sharing of bandwidth. More recently, Cisco Systems announced a new virtualization offering in conjunction with VMware ESX under the Unified Computing System (UCS) [3] umbrella. UCS is a proprietary solution with little details available, but the key components seem to be a Virtualized Ethernet Module (VEM) running under ESX on the hosts, and connected to a Cisco Systems Nexus switch managed by a Virtual Supervisor Module (VSM). The VEM does special proprietary tagging for packets (VN-tag) from all virtual NICs allowing the switching to happen on the external switch which can see each Virtual NIC as a port, and can ensure fairness by using bandwidth sharing amongst virtual machines. It also allows a centralized representation of all virtual machines view across all nodes, which allows expressing consistent link-wide policies for security, resource provisioning, and VM migration.

Unfortunately, the only way UCS has been deployed was through extremely tight integration with a particular OS virtualization flavor, namely the VMware ESX server. The VEM in fact replaces the whole Layer 2 of the networking stack, and does not function outside the ESX hypervisor. Cisco also offers a physical NIC which works with ESX and does the VN-tag insertion/removal and switching in the NIC hardware itself. Since VN-tag and the entire UCS architecture is proprietary and doesn't work with any existing switch, it will be a big hindrance to the adoption of the technology. Another shortcoming is the absence of any integration with the virtualization innovations on the NICs. This solution forces the traffic out of the machine on the wire, which causes additional overhead for traffic between local domains.

Other related work is currently taking place under the OpenFlow programmable switch project by Mckeown et. al. [10]. The core idea is to encourage switch vendors to develop a programmable switch that has a standard way of adding and deleting flows. The Crossbow approach of programming the NIC classifier based on VNICs MAC address and VLAN tags, combined with instantiating a complementary rule on an Openflow based switch in a open standard based mechanism, could go a long way in creating a truly virtualized layer 2 fabric.

7. CONCLUSIONS AND FUTURE WORK

The Crossbow architecture presented in this paper introduces a novel approach to achieve network virtualization by tackling the networking resources virtualization and fair bandwidth sharing on the host. Physical NIC resources are partitioned and assigned to VNICs, which are viewed by the rest of the system as full fledged data links. The dedicated virtualization lanes associated with the hardware resources assure proper data path separation, security, and fair sharing of bandwidth between virtual machines with no performance overhead. Dynamic polling on a per lane basis reduces the

per packet processing cost and enables the kernel to schedule the receive-side packet processing and easily implement bandwidth control.

Crossbow implements virtual switching between VNICs created on the same physical NIC, aggregation of physical NICs, or etherstub. VNICs and etherstubs are the main building blocks for the Virtual Wire concept. Virtual Wires enable the consolidation of multi-tier networks with complex topologies into a single or a small set of adequately provisioned virtualized hosts.

The core of the Crossbow architecture offering these features has been implemented and integrated in OpenSolaris and is available to any user at http://opensolaris.org.

Near-term work is around supporting IOV virtual functions as the IOV-capable NICs start emerging. A hybrid approach is being designed such that network operators can choose to assign a virtual function directly to a virtual machine or through the hypervisor, where security policies or virtual machine migration doesn't permit a direct mapping into the virtual machine.

Other future work includes dynamically provisioning hardware lanes based on real-time resource utilization and billing policies, as well as extending that model for provisioning virtual wire in the context of cloud computing.

8. REFERENCES

[1] A. Banerjee. Introducing uperf - an open source network performance measurement tool, 2008.

[2] P. Barham, B. Dragovic, K. Fraser, T. H. Steven Hand, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *19th ACM Symposium on Operating System Principles*, pages 164–177. ACM, 2003.

[3] Cisco. Unified comuting systems.

[4] P. henning Kamp and R. N. M. Watson. Jails: Confining the omnipotent root. In *In Proc. 2nd Intl. SANE Conference*, 2000.

[5] IEEE. 802.1ak multiple VLAN registration protocol.

[6] IEEE. 802.1Q GVRP VLAN registration protocol.

[7] Intel. Intel 82598 10GbE Ethernet Controller Open Source Datasheet, 2008.

[8] Intel. Intel VMDq technology, notes on software design support for Intel VMDq technology. White paper, March 2008.

[9] G. Liao, D. Guo, L. Bhuyan, and S. R. King. Software techniques to improve virtualized I/O performance on multi-core systems. In *4th ACM/IEEE Symposium on Architectures for Networking and Communication Systems, Nov 2008*. ACM, 2008.

[10] N. Mckeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. Whitepaper.

[11] A. Menon, A. L. Cox, and W. Zwaenepoel. Optimizing network virtualization in Xen. In USENIX, editor, *USENIX Annual Technical Conference, May 30–June 3, 2006, Boston, MA*, pages 15–28, 2006.

[12] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel. Diagnosing performance overheads in the Xen virtual machine environment. In ACM/USENIX, editor, *1st ACM/USENIX International Conference on Virtual Execution Environments, VEE 05 Conference*. ACM, 2005.

[13] Neterion. Neterion Xframe II 10 Gigabit Ethernet.

[14] D. Ongaro, A. Cox, and S. Rixner. Scheduling I/O in virtual machine monitors. In *Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 1–10. ACM, 2008.

[15] D. N. Padma Apparao, Srihari Makeneni. Characterization of network processing overheads in Xen. In *VTDC 06: 2nd International Workshop on Virtualization Technology in Distributed computing*. IEEE Computer Society, 2006.

[16] PCI-SIG. Single root I/O virtualization and sharing specification, 2007.

[17] PCI-SIG. MSI-X ECN for PCI express, 2008.

[18] D. Price and A. Tucker. Solaris zones: Operating system support for consolidating commercial workloads. In *18th Large Installation System Administration Conference*, pages 241–254. USENIX, 2004.

[19] S. Soltesz, H. Potzl, M. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: a scalable high-performance alternative to hypervisors. In *2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 275–287. ACM, 2007.

[20] Sun. Sun Multithreaded 10GbE (Gigabit Ethernet) Networking Cards, 2007.

[21] S. Tripathi, N. Droux, T. Srinivasan, K. Belgaied, and V. Iyer. Crossbow: A vertically integrated QoS stack. In *In Proceedings of the ACM SIGCOMM workshop WREN'09 (To appear)*, 2009.

[22] J. S. G. Venkitachalam and B. Lim. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proceedings of the 2001 USENIX Annual Technical Conference*, 2001.

[23] M. Zec. Implementing a clonable network stack in the FreeBSD kernel. In *In Proceedings of the USENIX 2003 Annual Technical Conference*, pages 137–150, 2003.

[24] J. Zhang, X. Li, and H. Guan. The optimization of Xen network virtualization. In *Computer Science and Software Engineering, 2008 International Conference on Persistent Link (OPAC)*, pages 431–436, 2008.