Crossbow Virtual Wire: Network in a Box

Sunay Tripathi, Nicolas Droux, Kais Belgaied, Shrikrishna Khare
Solaris Kernel Networking, Sun Microsystems, Inc.

Abstract

Project Crossbow in OpenSolaris is introducing new abstractions that provide virtual network interface cards (VNICs) and virtual switches that can have dedicated hardware resources and bandwidth assigned to them. Multiple VNICs can be assigned to OpenSolaris zones to create virtual network machines (VNM) that provide higher level networking functionality like virtual routing, virtual load balancing, and so on. These components can be combined to build an arbitrarily complex virtual network called virtual wire (vWire) which can span one or more physical machines. vWires on the same physical network can be VLAN-separated and support dynamic migration of virtual machines, which is an essential feature for hosting and cloud operators.

vWires can be reduced to a set of rules and objects that can be easily modified or replicated. This ability is useful for abstracting out the application from the hardware and the network, and thus considerably facilitates management and hardware upgrade.

The administrative model is simple yet powerful. It allows administrators to validate their network architecture, do performance and bottleneck analysis, and debug existing problems in physical networks by replicating them in virtual form within a box.

Keywords: Virtualization, Virtual Switches, VMs, Xen, Zones, QoS, Networking, Crossbow, vWire, VNICs, VNM.

1 Introduction

In recent years, virtualization[2][3][7] has become mainstream. It allows the consolidation of multiple services or hosts on smaller number of hardware nodes to gain significant savings in terms of power consumption, management overhead, and data-center cabling. Virtualization also provides the flexibility to quickly repartition computing resources and redeploy applications based on resource utilization and hardware availability. Recently these concepts have enabled cloud computing[6] to emerge as a new paradigm for the deployment of distributed applications in hosted data-centers.

The benefits of virtualization is not only in consolidation and capacity management. With virtualization, the operating environment can be abstracted[14][18] and decoupled from the underlying hardware and physical network topology. Such abstraction allows for easier deployment, management, and hardware upgrades. As such focus has shifted towards multiple forms of network virtualization that do not impose a performance penalty[23].

Project Crossbow in OpenSolaris offers high performance VNICs to meet the networking needs of a virtualized server that is sensitive to network latency and throughput. Crossbow leverages advances in the network interface cards (NICs) hardware by creating hardware based VNICs which offer significantly less performance penalties. The VNICs have configurable link speeds, dedicated CPUs, and can be assigned VLAN tags, priorities, and other data link properties. Crossbow also provides virtual switches to help build a fully virtualized layer-2 network.

The VNICs can be created over physical NICs, link aggregations for high availability, or pseudo NICs to allow the administrator to build virtual switches independently from any hardware. Networking functionality such as routing and packet filtering can be encapsulated in a virtual machine or zone with dedicated VNICs to form virtual network machines. These virtual network machines can be deployed on virtual networks to provide layer-2 and layer-3 networking services, replacing physical routers, firewalls, load balancers, and so on.

With all the virtualized components Crossbow provides, an administrator can build an arbitrarily complex virtualized network based on the application needs and decouple it from the underlying physical network. The resulting virtual network is called virtual wire. The vWire can be abstracted as a set of rules such as bandwidth limits, and objects such as VNICs and virtual switches, that can be combined, modified, or duplicated with ease and instantiated on any hardware. Crossbow allows migrating not just the virtual machine but entire virtualized network.

The functionality provided by Crossbow is part of the core OpenSolaris implementation, and does not require add-on products or packages.

In this paper we describe the main components of the Crossbow architecture from the perspective of a system and network administrator. We will introduce the new system and networking entities that are used for virtualizing the networking resources and for controlling the QoS at various granularities. We describe these entities with an emphasis on the simplified administration model by showing how they can be used as independent features, or as building blocks for the creation of vWires. In the examples section, we explore how Crossbow basic components can be used to build fully functional virtualized networks and new ways to do QoS. System administrators can also use the vWire to create a Network in a box to do performance, functionality, and bottleneck analysis.

2 Issues In Existing Models

The current methods of network virtualization are based on VLANs that are typically configured on the switches. This model is not very flexible if a VLAN tag is assigned to a virtual machine and the virtual machine needs to be migrated due to resource utilization needs. An administrator needs to manually add the virtual machine's VLAN tag to the switch port corresponding to the target machine. Protocols such as GVRP[13] and MVRP[17] are available for doing this dynamically. However, these protocols are not supported on a large number of switches.

The sharing of the common bandwidth between virtual machines also becomes an issue[9], as the current generation of switches offers fairness only on a per port basis. If the same port is shared by multiple virtual machines, any one of those virtual machines can monopolize usage of the underlying physical NIC resources and bandwidth. Host-based fairness or policy based sharing solutions impose significant performance penalties and are really complex to administer. They typically involve the creation of classes, the selection of queuing models, jitters, bursts, traffic selectors, and so on, all of which require an advanced knowledge of queuing theory.

Virtual networks that are created by using the existing VLANs and QoS mechanisms are prone to errors in the event of configuration changes or workload changes. The connectivity and performance testing is based on home grown solutions and requires expensive hardware based traffic analyzers. Often, there are heavy performance penalties and non-repeatable performance that depends on interactions with other virtual machines of different virtual networks.

This document will show how Crossbow can move VLAN separation and enforcement into the host and allow virtual machines to migrate without requiring changes to the physical network topology or switches. It will also show how VNICs can be associated with a link speed, CPUs, and NIC resources to efficiently and conveniently provide fair sharing of physical NICs. VNICs and virtual switches can be combined to build virtual networks which can be observed and analyzed by using advanced operating system tools such as DTrace.

3 Crossbow Virtualization Components

This section discusses the various Crossbow components that enable full virtualization, from virtualizing hardware resources such as NICs to building scalable vWire and network in a box.

3.1 Virtual NICs

When a host is virtualized, the virtual environment must provide virtual machines (VMs) connectivity to the network. One approach would be to dedicate one NIC to each virtual machine. While assigning dedicated NICs ensures the isolation of each VM's traffic from one another, this approach defeats one of the main purposes of virtualization, which is to reduce cost from the sharing of hardware. A more efficient and flexible option is to virtualize the hardware NICs themselves so that they can be shared among multiple VMs.

Crossbow provides the concept of the VNICs. A VNIC is created on top of a physical NIC, and multiple VNICs can share the same physical NIC. Each VNIC has a MAC address and appears to the system as any other NIC on the system. That is, VNICs can be configured from the IP stack directly, or they can be assigned to virtual machines or zones.

Crossbow can also assign dedicated hardware resources to VNICs to form *hardware lanes*. Most modern NIC hardware implementations offer hardware classification capabilities[10][20][12] which allow traffic for different MAC addresses, VLANs, or more generic traffic flows to be directed to groups of hardware rings or DMA channels. The Crossbow technology leverages these hardware capabilities by redirecting traffic to multiple VNICs in the hardware itself. The redistribution of traffic reduces network network virtualization overhead and provides better isolation between multiple VNICs that share the same underlying NIC.

In Crossbow VNICs are implemented by the OpenSolaris network stack as a combination of the virtualized MAC layer and a pseudo VNIC driver. The virtualized MAC layer interfaces with network device drivers under it, and provides a client interface for use by the network stack, VNICs, and other layered software components. The MAC layer also implements the virtual switching capabilities that are described in Section 3.3. The VNIC driver is a pseudo driver and works closely with the MAC layer to expose pseudo devices that can be managed by the rest of the OS as a regular NIC.

For best performance, the MAC layer provides a pass-through data-path for VNICs. This pass-through allows packets to be sent and received by VNICs clients without going through a bump-in-the-stack, and thus minimize the performance cost of virtualization. To assess the performance impact of VNICs, we measured the bi-directional throughput on a testbed consisting of 5 clients firing packets at a single receiver (quad-core, 2.8GHz, Intel-based machine) through a 10 Gigabit Ethernet switch. The measured performance of a VNIC with dedicated hardware lanes was the same as the performance of the physical NIC with no virtualization[24].

A side-effect of that architecture is that it is not possible to directly create VNICs over VNICs, although VNICs can be created on top of other VNICs indirectly from different OS instances.

Crossbow VNICs have their own dedicated MAC addresses and as such, they behave just like any other physical NIC in the system. If assigned to a virtual machine or zone, the VNIC enables that virtual machine to be reachable just like any other node in the network.

There are multiple ways to assign a MAC address to a VNIC:

Factory MAC address: some modern NICs such as Sun's 10 Gigabit Ethernet adapter[20] come from the factory with multiple MAC addresses values allocated from the vendor's MAC address organizationally unique identifier (OUI). VNICs can be assigned one of these MAC addresses if they are provided by the underlying NIC.

Random MAC address: A random MAC address can be assigned to a VNIC. The administrator can either specify a fixed prefix or use the default prefix. Crossbow will randomly generate the least significant bits of the address. Note that after a random MAC address is associated with a VNIC, Crossbow makes that association persistent across reboots of the host OS. To avoid conflicts between randomly generated MAC addresses and those of physical NICs, the default prefix uses an IEEE OUI with the local bit set. There is currently no guarantee that a randomly generated MAC address does not conflict with other MAC addresses on the network. This functionality will be delivered as part of future work.

Administratively set MAC Address: If the administrator manages the set of MAC addresses of the virtual machines or zones, he/she can supply the complete MAC address value to be assigned to a VNIC.

VNICs are managed by `dladm(1M)`, which is the command used to manage data links on OpenSolaris. Section 4.1.1 describes in details VNIC administration with the `dladm(1M)` command. A VNIC appears to the rest of the system as a regular physical NIC. It can be managed by other existing built-in tools such as `ifconfig(1M)`, or by third-party management tools.

VNICs have their own statistics to allow real time and historical analysis of network traffic that traverse them. Section 4.3 describes VNIC statistics and their analysis.

Last but not least, the traffic going through VNICs can be observed by existing tools such as `snoop(1M)`. Capturing packets going through VNICs is similar to observing the traffic on a physical switch port. That is, for a particular VNIC, only the broadcast and multicast traffic for the VLAN IDs associated with the VNIC, as well as the unicast traffic for the VNIC MAC address, are visible for observation.

3.2 Configurable Link Speeds

Transport protocol implementations will attempt to use the bandwidth that is made available by the underlying NIC[4]. Similarly, multiple VNICs defined on top of the same underlying NIC share the bandwidth of that NIC. Each VNIC will attempt to use as much as it can from the link's bandwidth. Various undesirable behaviors can ensue from this situation:

- A transport or a service can be an active offender – Some transport protocols are more aggressive than others. For example a UDP sender will not throttle its transmission rate even if the receiver cannot keep up with the received traffic. On the other hand, protocols like TCP will slow the sender down if needed. Such differences in behavior can lead to a VNIC for UDP traffic consuming more of the underlying bandwidth than other VNICs that are used for TCP.

- A client virtual machine can be a passive target of an external attack – In a virtualized setup where a hardware node is used to host virtual machines of different customers, one or more of those customers can become a victim of a denial of service attack[15][16]. The virtual machine for one customer can end up using most of the link's capacity, effectively diminishing the performance of all the virtual machines that share the same NIC.

- Some VMs may have different bandwidth needs than others – The bandwidth of a NIC should be partitioned between VNICs to satisfy the requirements of the VMs. In some instances customers could be charged a premium if a larger share of the bandwidth is allocated to them. An uncontrolled or even egalitarian sharing of the resources might not necessarily be the desired behavior.

With the `dladm(1M)` command, Crossbow allows the link speed of data links to be specified through link properties. Configuring the link speed is the equivalent of setting a maximum bandwidth limit on the data link. This property can be configured explicitly by the administrator, or it can be set from the host OS of a virtualized environment when the VNIC for a virtual machine is created, as shown in Section 4.2 below.

3.3 Virtual Switching

When multiple VNICs are created on top of a physical NIC, the MAC layer automatically creates a virtual switch on top of that NIC. All VNICs created on top of the physical NIC are connected to that virtual switch. The virtual switch provides the same semantics as a physical switch. Figure 1 shows the mapping between physical NICs and switches and their virtual equivalent in Crossbow. Note that multiple VNICs can be created on different physical NICs. In such cases, each physical NIC will be assigned its own virtual switch. Virtual switches are independent, and there are no data paths between them by default.

3.3.1 Outbound Packet Processing

When a packet is sent by a client of a VNIC, the virtual switch will classify the packet based on its destination MAC address. The following actions are taken depending on the result of that classification:

- If the destination MAC address matches the MAC address of another VNIC on top of the same physical NIC, the packet is passed directly to that VNIC without leaving the host.

- If the MAC address is a broadcast MAC address, a copy is sent to all VNICs created on top of the same physical NIC, and a copy is sent on the wire through the underlying NIC.

- If the MAC address is a multicast MAC address, a copy of the packet is sent to all VNICs which joined the corresponding MAC multicast group, and a copy is sent through the underlying NIC. The MAC virtual switch maintain a list of multicast membership for this purpose.

- If MAC destination is unknown, i.e. there is no entry for the MAC address in the layer-2 classification table of the virtual switch, the packet is passed down to the underlying physical NIC for transmission on the wire.

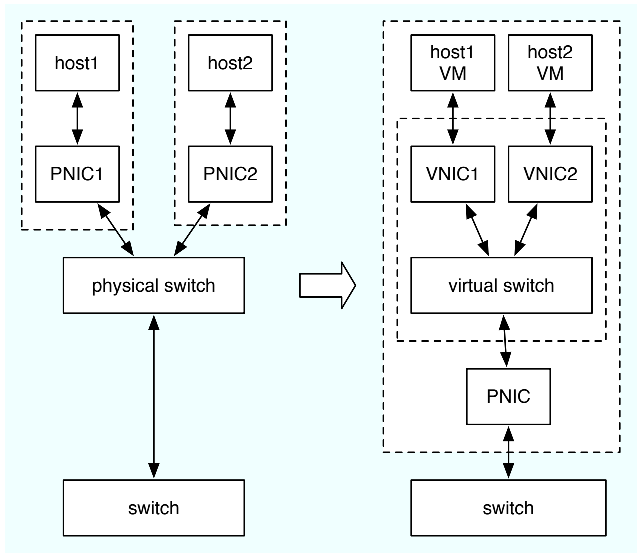

Figure 1: Mapping between physical and virtual switches

3.3.2 Inbound Packet Processing

Packets received off the wire are first classified by the NIC hardware according to the destination MAC address of the packet. If there is a match after hardware classification, the NIC hardware deposits the packet in one of the hardware rings associated with the MAC address. The MAC address and VNIC that are associated with that hardware ring is known to the host. Thus, when the host picks up the packet from that ring, it can deliver the packet to the correct VNIC network stack or virtual machine.

If the hardware classifier cannot find a dedicated hardware ring for the destination MAC address of the incoming packet, it deposits the packet in one of the dedicated hardware default receive rings. The MAC layer performs software classification on the packets received from these default rings to find the destination VNIC.

3.4 Etherstubs

We have seen in Section 3.3 that Crossbow creates a virtual switch between the VNICs sharing the same underlying physical NIC. As an alternative, VNICs can also be created on top of *etherstubs* to create virtual switches which are independent of any hardware. Etherstubs are pseudo ethernet NICs and are managed by the system administrator. After an etherstub is created, it can be used instead of a physical NIC to create VNICs. The MAC layer will then perform virtual switching between the VNICs which share the same underlying etherstub.

Etherstubs and the MAC layer virtual switching allow users to create virtual switches which are independent from physical NICs. Whether the virtual switch is implicitly created over a link (physical NIC or an aggregation), or explicitly built by an etherstub, all VNICs sharing the same virtual switch are connected and can communicate with one another. Conversely, VNICs that are not members of the same virtual switch are isolated from each other. Figure 2 shows how virtual switching can be used between VNICs with both physical NICs and etherstubs.

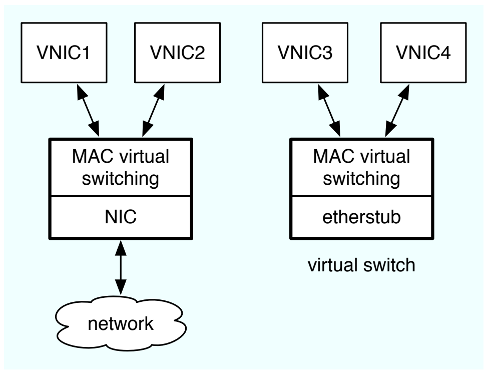

Figure 2: Virtual switching with physical NICs and etherstubs

Multiple etherstubs can be created to construct multiple virtual switches which can be combined to form flexible virtual networks. Section 5.2 shows an example of such an architecture.

3.5 VLANs

IEEE 802.1 VLANs can be used to build isolated virtual LANs sharing the same underlying physical layer-2 network infrastructure. Each VLAN is associated with a VLAN tag and defines its own broadcast domain. Hardware switches allow the traffic of different VLANs to be separated, and to associate switch ports with specific VLAN tags.

The Crossbow virtual switching is VLAN-aware and thus allows VLAN separation to extend to virtual switches and VNICs. VNICs can be associated with a *VLAN identifier*, or *VID*, which is used along with the MAC address to classify traffic to VNICs. As it is the case of physical switches, the Crossbow virtual switch also implements per-VLAN broadcast domains. In other words, tagged broadcast frames will be delivered only to the VNICs that match the VLAN tag. From the perspectives of efficiency and security, the Crossbow VLAN implementation provides two important features: it prevents the unnecessary duplication of frames and it ensures that no leakage of frames to the wrong VLAN is occurring.

Control of the VLAN handling is deliberately kept to the MAC layer of the host OS (or global zone when applicable). When a VNIC is used by a guest VM, the VM can only send and receive untagged traffic. The host's MAC layer inserts or strips the VLAN tag transparently. It also ensures that the VM does not attempt to send tagged packets. Thus, the VM cannot send packets on a VLAN to which it does not belong.

3.6 High Availability and VNICs

In order to provide highly available network connectivity, OpenSolaris supports availability at layer-2 and layer-3 by means of link aggregations and IPMP, respectively.

3.6.1 Layer-2: IEEE 802.3ad Link Aggregation

Link aggregations are formed by grouping multiple NICs in a single pseudo NIC. Multiple connections are spread through the NICs of the aggregation. Ports are taken out of the aggregation if they are misconfigured or fail unexpectedly. Failure detection is achieved by monitoring the link state of aggregated NICs or by exchanging Link Aggregation Control Protocol (LACP) control messages at regular intervals.

In OpenSolaris, link aggregations are managed by using `dladm(1M)` and implemented by a pseudo driver which registers with the system a pseudo NIC for each configured link aggregation. Each instance of the pseudo driver behaves like any other NIC on the system. As such, the pseudo driver allows VNICs to be created on top of link aggregations in the same manner that VNICs can be created on top of physical NICs or etherstubs. Figure 3 shows how two physical NICs can be aggregated, virtualized, and shared transparently by two guest domains.

The IEEE link aggregation standard assumes that an aggregation is built between two entities on the network. Typically these entities are switches and hosts. Unfortunately, this standard does not allow an aggregation to connect one host to multiple switches, which is a desirable configuration as a measure against possible switch failure. Some switch vendors have provided extensions called *switch stacking* that allow an aggregation to span multiple switches. These extensions are transparent to the peers that are connected to the switch stack.

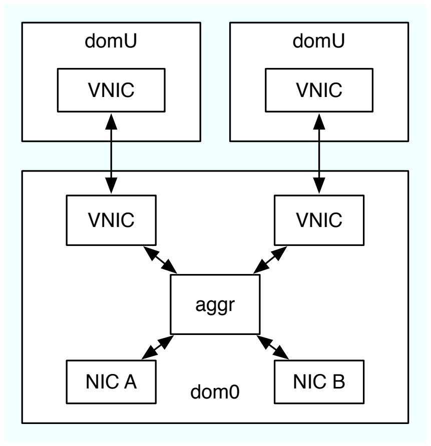

Figure 3: Using link aggregation to provide high-availability and increased throughput to VNICs

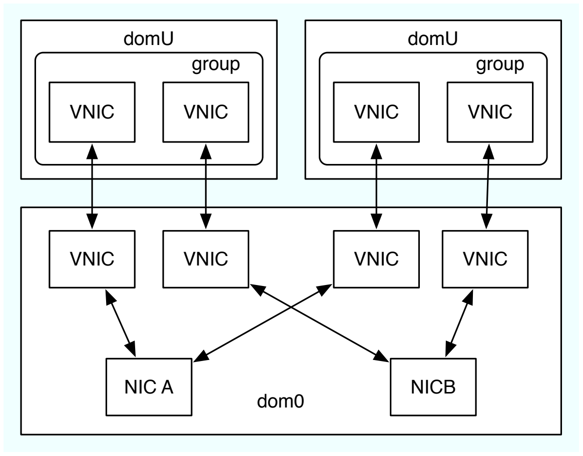

Figure 4: Using IP multipathing from virtual machines for high-availability

3.6.2 Layer-3: IP Multipathing

IP Multipathing, or IPMP[19], is a layer-3 high availability feature. It allows multiple IP interfaces to be grouped together, and provides load spreading and failover across members of the group. IPMP provides link-based detection failure, and probe-based detection failure.

Since IPMP is at layer-3 above NIC virtualization, VNICs cannot be created on IPMP groups and IPMP high availability cannot be provided transparently to virtual machines. Instead, VNICs can be created on each physical NIC, and VNICs can be grouped within virtual machines. Figure 4 shows how two NICs can be virtualized and grouped within virtual machines. IPMP groups are managed by using the `ifconfig(1M)` IP configuration tool.

Note that link aggregation and IPMP can be combined. For example, link aggregations can be used to group multiple NICs connected to the same switches, and IPMP can be used to group multiple link aggregations.

3.7 Virtual Network Machines

Virtual NICs and virtual switching constructs are the building blocks that allow more complex virtual networking topologies to be built within a host. The functionality needed to implement typical networking devices on a network, such as routers or firewalls, exists in modern operating systems like OpenSolaris. Networking devices can be therefore encapsulated within virtual machines or OpenSolaris zones.

An OpenSolaris zone is a lightweight virtualization architecture where the zone provides its own application environment that is isolated from other zones[21]. Each zone can be associated with a set of CPUs, data links such as VNIC, memory cap, and so on. Zones share the same kernel but each zone can have its own IP network stack. This feature avoids overheads that are typically associated with hypervisors. Because of their low overhead, small memory footprint, and specific functionality that does not require a full separate OS instance, zones are particularly suited to implement virtual network devices.

Virtual network machines refer to virtual machines or zones which are dedicated to implementing specific network functions. VNMs can be connected by assigning them VNICs and connecting these VNICs to virtual switches. Several types of network functions can be implemented, such as routers, firewalls, load balancers, and bridges. With Crossbow, essentially any layer-2 or layer-3 network can be virtualized within a single host.

3.8 Traffic Flows

Crossbow flows allow bandwidth limits, CPUs, and priorities to be associated with a subset of the network traffic that traverses a NIC, link aggregation, or VNIC. Flow attributes describe the traffic that is associated with the flows. Attributes consist of information such as IP addresses, well known port numbers, protocol types, and so on.

Crossbow flows span the whole network stack from the NIC hardware to sockets, and are associated with their own kernel threads and available hardware resources. Their specific associations make flows distinct from one another. Consequently, after hardware classification of incoming traffic is performed, traffic processing of flows can be scheduled independently from each other as well. With a setup that uses Crossbow, flows are better isolated, the task of classification is assumed by the hardware, and the network stack can control the arrival of traffic into the host on a per-flow basis.

Flows also maintain their own statistics to allow an administrator to track real-time statistics and usage history not only of individual data links as a whole but also of specific types of traffic the host receives or sends. Traffic flows are described in more detail in[25].

4 Ease of Management

Crossbow provides management tools that are easy to use to create VNICs, connect VNICs by using virtual switches to build vWires, and configure networking resources for these VNICs' dedicated use. In addition, statistics on traffic flows, both real time and historical, provide the administrator the ability to monitor traffic at a deeper granularity and thus better allocate networking resources. This section describes the Crossbow tools to perform these tasks.

4.1 Managing vWire

The vWire building blocks are managed through the `dladm(1M)` command, the OpenSolaris data-link management utility. This section shows how the `dladm(1M)` tool can be used to perform the following:

- Manage VNICs.
- Combine VNICs with etherstubs to build virtual networks.
- Combine VNICs with link aggregations to provide high availability and increased throughput to virtual machines and zones.

4.1.1 NIC Virtualization

As seen in Section 3.1, VNICs can be used to virtualize a data link. A VNIC is easily created with the `dladm(1M) create-vnic` subcommand. The following example shows the creation of a VNIC called `vnic100` on top of the physical NIC `e1000g4`.

```
# dladm create-vnic -l e1000g4 vnic100
```

In this case the administrator lets the system determine the MAC address to be associated with the VNIC. Users can choose any administratively meaningful name for the data links (NICs, VNICs, aggregations, etherstubs, and so on) as long as the name ends with a numeral. The `dladm(1M) show-vnic` subcommand can be used to display the VNIC configuration. For example:

```
# dladm show-vnic -o LINK,OVER
LINK          OVER
vnic100       e1000g4
# dladm show-vnic -o LINK,MACADDRESS
LINK          MACADDRESS
vnic100       2:8:20:36:ed:5
# dladm show-vnic -o LINK,OVER,MACADDRESS
LINK          OVER          MACADDRESS
vnic100       e1000g4       2:8:20:36:ed:5
```

The previous example shows how the `-o` option can be used to specify the fields to be displayed for each VNIC. If the `-o` option is omitted, then all attributes of the VNICs will be displayed.

VNIC attributes such as the specified MAC address to be associated with the VNIC can be specified by the user as additional options of `create-vnic`. The `dladm(1M) delete-vnic` subcommand can be used to delete previously created VNICs from the system. Of course, multiple VNICs can be created on top of the same physical NIC.

After a VNIC is created, it appears to the rest of the system as a regular data link and therefore can be managed in the same way as other NICs. It can be plumbed by the network stack directly as shown below, or assigned to a virtual machine as shown in Sections 4.2.1 and 4.2.2.

```
# ifconfig vnic100 plumb
# ifconfig vnic100 inet 10.20.20.1/24 up
# ifconfig vnic100
vnic100: flags=1000843<UP,BROADCAST,...
      inet 10.20.20.1 netmask ffffff00
      broadcast 10.20.20.255
      ether 2:8:20:36:ed:5
```

4.1.2 Etherstubs

Etherstubs are constructs that can be used to build virtual switches which are completely independent from physical NICs (see Section 3.4.) An etherstub can be used instead of a physical NIC to create VNICs. The VNICs sharing the same etherstub then appear to be connected through a virtual switch.

In the following example, an etherstub `vswitch0` is created, and then used to create three VNICs: `vnic0`, `vnic1`, and `vnic2`.

```
# dladm create-etherstub vswitch0
# dladm create-vnic -l vswitch0 vnic0
# dladm create-vnic -l vswitch0 vnic1
# dladm create-vnic -l vswitch0 vnic2
```

4.1.3 VLANs

Section 3.5 described how VLANs can be seamlessly integrated in the virtualization environment and used to create multiple virtual networks on the same underlying physical infrastructure. A VLAN can be easily associated with a VNIC during its creation.

```
# dladm create-vnic -l e1000g0 \
-v 200 vlan200vnic0
# dladm create-vnic -l e1000g0 \
-v 200 vlan200vnic1
# dladm create-vnic -l e1000g0 \
-v 300 vlan300vnic0

# dladm show-vnic -o LINK,MACADDRESS,VID
LINK          MACADDRESS           VID
vlan200vnic0 2:8:20:d5:38:7        200
vlan200vnic1 2:8:20:69:8f:ab       200
vlan300vnic0 2:8:20:3a:79:3a       300
```

As shown in the previous example, multiple VNICs can be created on top of the same physical NIC or etherstub with the same VID. In this case, the MAC layer virtual switching isolates these VLANs from each other, but will allow VNICs with the same VID to communicate together as if they were connected through a switch.

4.1.4 Link Aggregation

Link aggregations are also managed through the `dladm(1M)` utility. A link aggregation can be easily created as shown in the example below where an aggregation called `aggr0` consisting of two physical NICs, `e1000g2` and `e1000g3` is created.

```
# dladm create-aggr -l e1000g2 \
-l e1000g3 aggr0
```

The resulting `aggr0` is a regular data link on the system. It can be configured using `ifconfig(1M)`, or it can be used to create VNICs which are then assigned to zones or virtual machines. In the example below, two VNICs are created on top of `aggr0`:

```
# dladm create-vnic -l aggr0 vnic500
# dladm create-vnic -l aggr0 vnic501
```

4.1.5 Management Library

The `dladm(1M)` command is a thin CLI above the OpenSolaris data link management library libdladm. The bulk of the work is done by the library, while the command line tool implements the parsing and formatting needed. The libdladm management library is also used by other management tools, agents, and utilities.

4.1.6 Network Flows

Crossbow provides a new command `flowadm(1M)` to configure flows. As described in Section 3.8, flows can be used from vWire to control and measure bandwidth usage of finer grain traffic. The `flowadm(1M)` command takes as its arguments a data link name, traffic criteria, priority, and desired bandwidth. Traffic criteria can be specific protocols, protocol ports, or local or remote IP addresses.

For example, a flow to match all UDP traffic passing through NIC `ixgbe0` can be created as follows:

```
# flowadm add-flow -l ixgbe0\
-a transport=udp udp-flow
```

Each flow has associated properties specified by the `-p` option. These properties can be used to define the maximum bandwidth or priority for a flow. Properties of existing flows can be changed without impacting the flow's defined criteria. By default, `udp-flow` uses the bandwidth of the underlying NIC, which in the example is 10 Gb/s. To change the bandwidth of `udp-flow` to 3 Gb/s, issue the following command:

```
# flowadm set-flowprop -p maxbw=3G \
udp-flow
```

If no speed unit is specified, the `maxbw` property unit is assumed to be in megabits per second (Mb/s). Additionally, the `flowadm(1M) show-flow` and `show-flowprop` subcommands can be used to display flow configuration and properties respectively. Flows can be deleted using the `flowadm(1M) remove-flow` subcommand.

4.2 Resource partitioning and QoS

Configuring QoS policies often tends to be laborious. For example, a typical policy might be to limit TCP traffic to use a bandwidth of 1000 Mb/s. However, configuring such a policy by using IPQoS in Solaris 10[19] or tc[5] in Linux entails several complex steps such as defining queuing disciplines, classes, filter rules, and the relationships among all of them.

The subsections that follow use real life scenarios to illustrate how Crossbow vastly simplifies QoS configuration.

4.2.1 Zones

With Crossbow, limiting bandwidth for a zone is simple to perform. One just needs to create a virtual NIC with the desired bandwidth and assign it to the zone. For example, to limit the bandwidth of zone `zone1` to 100 Mb/s, first create a VNIC with the desired bandwidth:

```
# dladm create-vnic -p maxbw=100 \
-l e1000g0 vnic1
```

When the zone is created, it can be given `vnic1` as its network interface:

```
# zonecfg -z zone1
...
zonecfg:zone1> add net
zonecfg:zone1:net> set physical=vnic1
zonecfg:zone1:net> end
...
```

Any traffic sent and received zone1 through vnic1 will be limited to 100 Mb/s. The configuration steps are a one time exercise. The configuration will be persistent across the zone or the operating system reboot. Changing the bandwidth limit at a later time can be achieved by setting maxbw property of that VNIC to the new value. Thus, to change bandwidth of zone1 to 200 Mb/s, use the following command syntax:

```
# dladm set-linkprop -p maxbw=200 vnic1
```

One can query the VNIC property zone to determine if the VNIC is assigned to any zone. Using the previous example, zone under the VALUE field indicates that vnic1 is a link that is being used by zone1.

```
# dladm show-linkprop -p zone vnic1
LINK          PROPERTY         PERM VALUE
vnic1         zone             rw   zone1
```

Plans are currently under consideration to configure zones' VNICs and their bandwidth limits directly by using zonecfg(1M). Thus, VNICs with specific property values can be created automatically when the zones are booted.

4.2.2 Xen

When OpenSolaris is used as dom0 (host OS), Crossbow provides a simple mechanism to assign bandwidth limits to domUs (VM guests). The configuration process is similar to configuring bandwidth limits for zones. A VNIC is created with the desired bandwidth limit, and then supplied as an argument during domU creation. The domU could be running OpenSolaris, Solaris 10, Linux, Windows, or any other Xen supported guest. This process is independent of the choice of the domU. The procedure is explained in detail as follows:

When a Xen domU is created, Crossbow implicitly creates a VNIC and assigns it to the domU. To enforce a bandwidth limit for a domU, first, explicitly create a VNIC and assign it to domU during creation. Then, set the bandwidth limit for the Xen domU by setting the maxbw property of the VNIC.

For example, to limit the bandwidth of domU guest1 to 300Mb/s, the VNIC with the given bandwidth is first created:

```
# dladm create-vnic -p maxbw=300 \
-l e1000g0 vnic1
```

Then, to assign the newly configured VNIC to the Xen domU as its network interface, include the following in the domU's template.xml configuration file. Use the dladm(1M) show-vnic subcommand to display the MAC address of vnic1.

```
<interface type='bridge'>
<source bridge='vnic1'/>
<mac address='vnic1's mac address/>
<script path='vif-dedicated'/>
</interface>
```

Finally, the domU is created as follows:

```
# virsh create template.xml
```

Any traffic sent and received by the guest domain through vnic1 will be limited to 300 Mb/s. As with zones, the bandwidth can be changed at a later time by setting the maxbw property to the new value.

Plans are under consideration to configure bandwidth limit for Xen domUs by using Xen configuration tools such as xm(1M) and virt-install(1M). For example, the virsh-attach interface command will take the maximum bandwidth as an optional argument. The specific bandwidth limit is then automatically applied to the implicitly created VNIC when the domain is booted.

When using Linux as dom0, bandwidth control on guests can be configured as follows:[1]

1. Associate a queuing discipline with a network interface (tc qdisc).

2. Define classes with the desired bandwidth within this queuing discipline (tc class).

3. Using the IP address of the guest OS's interface, define a rule to classify an outgoing packet into one of the defined classes (tc filter).

For example, the following set of commands issued from dom0, would set bandwidth limits of 200 Mb/s and 300 Mb/s for each one of the domU instances, and reserve the remaining 500 Mb/s for dom0' use[8].

```
# tc qdisc add dev peth0 \
root handle 1: htb default 99

# tc class add dev peth0 \
parent 1: classid 1:1 htb rate 1000mbps \
burst 15k

# tc class add dev peth0 parent 1:1 \
```

[1]At the time of writing this paper, the latest Fedora release that could host Xen guests was Fedora 8 (Fedora 9 and Fedora 10 cannot host Xen guests). It supports a vif parameter 'rate'to control bandwidth limit. However, due to a bug (RedHat bug id 432411), we could not evaluate that feature.

```
classid 1:13 htb rate 200mbps burst 15k
# tc class add dev peth0 parent 1:1 \
classid 1:14 htb rate 300mbps burst 15k
# tc class add dev peth0 parent 1:1 \
classid 1:99 htb rate 500mbps burst 15k


# iptables -t mangle -A POSTROUTING \
-p tcp -s 192.168.1.103 -j CLASSIFY \
--set-class 1:13
# iptables -t mangle -A POSTROUTING \
-p tcp -s 192.168.1.104 -j CLASSIFY \
--set-class 1:14
# iptables -t mangle -A POSTROUTING \
-p tcp -s 192.168.1.111 -j CLASSIFY \
--set-class 1:21
```

Note that the previous approach does not work well when domUs obtain IP addresses by using DHCP. Moreover, domU users can circumvent the bandwidth limit enforcement by changing their IP address.

4.2.3 Traffic Flows

In the previous example, we restricted all traffic passing through a Xen domU to 300 Mb/s. Suppose that we further want to partition the available 300 Mb/s bandwidth as follows: 100 Mb/s for all TCP traffic and the remaining 200 Mb/s for all other traffic. Crossbow can achieve this configuration by using flows:

```
# flowadm add-flow -p maxbw=100 \
-a transport=tcp -l vnic1 tcp-flow1
```

The concept of flows is applicable to non-virtualized context as well. For example, a physical NIC can be specified instead of a VNIC. Thus, Crossbow provides a simple yet powerful way to administer bandwidth.

In contrast, configuring policies with `iproute(8)` and `tc(8)` on Linux typically involves several steps, For example:

```
# tc qdisc add dev eth4 handle ffff: \
ingress

# tc filter add dev eth4 parent ffff: \
protocol ip prio 20 \
u32 match ip protocol 6 0xff \
police rate 1Gbit buffer 1M drop \
flowid :1

# tc qdisc add dev eth4 root \
handle 1:0 cbq bandwidth 10Gbit \
avpkt 1000 cell 8

# tc class add dev eth4 parent 1:0 \
classid 1:1 cbq bandwidth 10Gbit \
rate 10Gbit prio 8 \
allot 1514 cell 8 maxburst 20 \
avpkt 1000 bounded

# tc class add dev eth4 parent 1:1 \
classid 1:3 cbq bandwidth 10Gbit \
rate 1Gbit weight 0.1Gbit prio 5 \
allot 1514 cell 8 maxburst 20 \
avpkt 1000

# tc class add dev eth4 parent 1:1
classid 1:4 cbq bandwidth 10Gbit \
rate 9Gbit weight 0.9Gbit prio 5 \
allot 1514 cell 8 maxburst 20 \
avpkt 1000

# tc qdisc add dev eth4 parent 1:3 \
handle 30: pfifo

# tc qdisc add dev eth4 parent 1:4 \
handle 40: pfifo

# tc filter add dev eth4 parent 1:0 \
protocol ip prio 1 u32 match ip \
protocol 6 0xff flowid 1:3
```

4.2.4 Flow Tradeoffs

The Crossbow design has traded off richness of flow attributes for simplicity and performance. Crossbow has departed from the traditional ways to specify QoS that consists of the following steps:

- Definition of classes of services
- Addition of rules similar to those of packet filtering
- Description of the packets that are assigned to each class

Instead, a flow is created by specifying its defining attributes that constitute as the common criteria that packets should match in order to belong to that flow. Resource controls policies, such as bandwidth constraints, priority and CPUs are viewed as mutable properties that can be allotted to flows at creation time and can be modified later.

Although flows can be created based on different attributes such as IP addresses, subnets, transport, DSCP marking, and port number, flows are defined based only on one attribute at a time, not on a combination of multiple attributes. Furthermore, only non overlapping flows are allowed to co-exist over a data link. Any attempt to create a flow that conflicts with an existing one fails. This apparent limitation provides the advantage of keeping the rule set that describes the flows inside the system unambiguous and order independent. A lookup for the flow that matches a packet will always find the same flow, regardless of the presence of other flows or the order in which they were added.

4.3 Monitoring Network Statistics

Crossbow also provides a rich set of statistics for gaining better insight into the behavior of the system. This section describes the tools provided to observe these statistics, and concludes with an example scenario to illustrate how these tools can be combined with other commands to diagnose and resolve a performance issue.

4.3.1 dlstat(1M) and flowstat(1M)

Crossbow statistics are provided on a per flow or data link basis. They provide information such as the count of packets received by polling and by interrupts, hardware and software packet drops, distribution of load across hardware lanes and so on. These statistics help to identify performance bottlenecks.

The current interface provides counts over a certain interval. Future improvements will provide more sophisticated aggregate level statistics such as percentage of polled packets, minimum, maximum, and average queue lengths over a specified time interval, and so on.

Crossbow introduces `dlstat(1m)` to print dynamic traffic statistics for links. For example, the following command prints the aggregate statistics for `vnic1`:

```
# dlstat vnic1
LINK   IPKTS IBYTES OPKTS  OBYTES
vnic1   9.9M   2.3G  4.8M    0.3G
```

To observe traffic exchange at 5-second interval, use the following:

```
# dlstat -i 5 vnic1
LINK   IPKTS IBYTES OPKTS  OBYTES
vnic1   1.5M   0.3G  0.6M   46.9M
vnic1   2.2M   0.5G  1.1M   73.3M
  .       .      .     .       .
```

Apart from dynamic statistics, `dlstat(1M)` also supports off-line viewing and analysis of statistics. `acctadm(1m)` is used to enable logging network statistics to a specific log file. The `dlstat(1M) -u` sub-option can then operate on the log file to extract historical network statistics. For example, the following command will extract network statistics for `vnic1` from the specified time range from `logfile`.

```
# dlstat -u -f logfile \
-s D1,shh:smm:sss -e D1,ehh:emm:ess vnic1
```

The output, if generated using `-F gnuplot` option, could be directly fed to `gnuplot` to draw graphical usage information for `vnic1`.

To analyze detailed receiver side statistics such as poll and interrupt packet counts as well as hardware and software drops, do the following:

```
# dlstat -r
LINK     IBYTES INTRS POLLS HDRPS
e1000g0    2.1M 22.3K  78.0   0.0
ixgbe0    13.6G  0.8K 10.7M   0.0
vnic1     13.6G  0.8K 10.7M   0.0
```

To also analyze per hardware lane statistics, append the `-L` option to the previous command. For example, the following will show per hardware lane statistics for each hardware lane that belongs to `ixgbe0`.

```
# dlstat -r -L ixgbe0
LINK:LNE LTYP  USEDBY IBYTES INTRS POLLS
ixgbe0:0 slne  ixgbe0  13.6G  0.8K   0.0
ixgbe0:1 hlne  ixgbe0  13.1G  0.8K 10.2M
    .       .       .      .     .     .
    .       .       .      .     .     .
ixgbe0:7 hlne  ixgbe0  13.4G  0.8K 10.5M
```

While `dlstat(1M)` operates on data links, `flowsat(1M)` is used for querying network statistics for flows. For example, to display `tcp-flow`'s network traffic statistics, do the following:

```
# flowstat tcp-flow
FLOW     LINK  IBYTES OPKTS  OBYTES
tcp-flow vnic1   2.3G  4.8M    0.3G
```

Like `dlstat(1M)`, `flowstat(1M)` also supports logging network statistics by using the `-u` sub-option.

Both inbound and outbound traffic statistics are shown by `dlstat(1M)` and and `flowstat(1M)`. The bandwidth limits apply to the combined bidirectional traffic, which is the sum of incoming and outgoing packets over time. Although we can observe the statistics for each direction, we currently can't set a different limit on each.

4.3.2 Example: Diagnosing a Scalability Issue

Consider a multi-processor system under heavy network load that uses the NIC `ixgbe0` and whose receiver side network performance needs improvement. Suppose that the output of `dlstat -r -L` is satisfactory. That is, after listing per-hardware lane packet and byte counts as well as poll and interrupt counts, you observe that traffic is evenly distributed across hardware lanes and that 95% of packets are delivered by polling. You can then check CPU utilization as follows:

- `dlstat -r -F ixgbe0` gives the breakdown of which CPUs are currently being used to process packets received by `ixgbe0`.
- `dladm show-linkprop -p cpus ixgbe0` displays the list of CPUs associated with the data link.
- `mpstat(1M)` provides information about the utilization of each CPU that is associated with `ixgbe0`.

Suppose that the data indicates that all the CPUs that are currently assigned to `ixgbe0` for packet processing are fully utilized while other CPUs in the system are at an idle or near-idle state. To dedicate a new list of CPUs for `ixgbe0`'s use, the following command syntax is used:

```
# dladm set-linkprop \
-p cpus=<list of cpus> ixgbe0
```

5 Virtual Wire: Network in a Box

We have described so far the major components needed for achieving network virtualization using convenient and intuitive tools. We then showed how bandwidth and computing resources can be awarded and controlled at a fine granularity to data links and VNMs. We can now use the VNMs, VNICs, etherstubs, along with the virtual switching and resource control capabilities as the building blocks to construct fully functional vWires of arbitrarily complex topologies in a single or small set of machines. The three scenarios below are examples of vWires used for consolidation of subnet and enterprise networks and for planning of horizontal scaling.

5.1 Example 1 – Seamlessly Consolidating Multiple Subnets

This example illustrates the high availability and elasticity features of vWires. It shows how two subnets can be consolidated together without any change to the IP configuration of the machines. It also shows how this consolidation not only reduces the cost but also increases the availability of all existing services. Figure 5 represents the two independent subnets. To emphasize the elasticity point, the subnets use the same internal IP addresses.

The consolidation must meet the following two requirements:

- Existing IP addresses must be retained. Many services in the network such as firewalls, proxies, directory services, kerberos, and so on depend on IP addresses. Reassigning IP addresses during consolidation risks breaking down these services and therefore must be avoided.

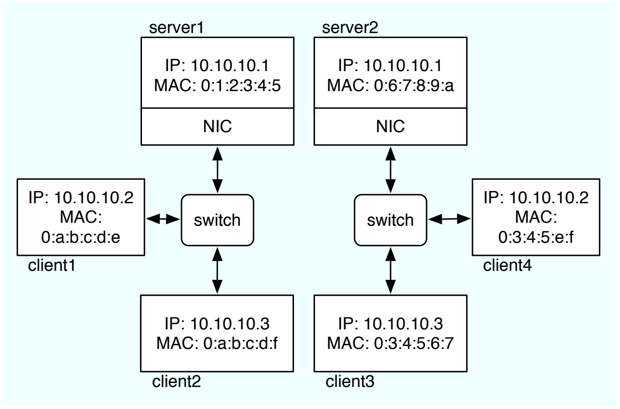

Figure 5: Example 1 – two separate physical subnets

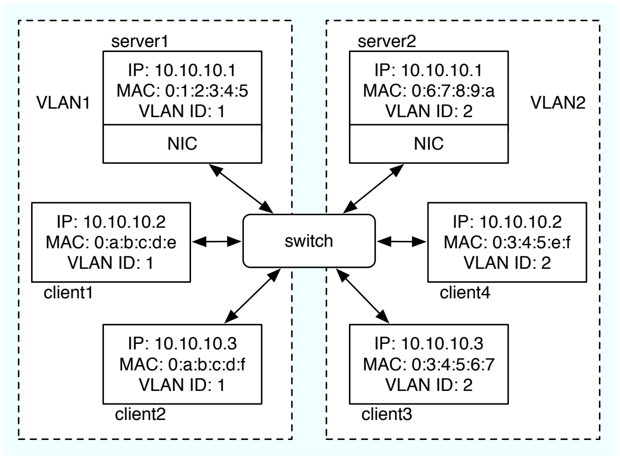

Figure 6: Example 1 – two VLANs sharing a physical network

- The consolidation must preserve the separation of traffic from the different subnets on the wire.

The traditional way to consolidate the two subnets on the same physical network would be to assign each subnet a VLAN ID, and then configure the switch ports with the appropriate VLAN IDs of the subnet. Finally, each machine is connected to the correct port. A VLAN-based network consolidation is represented in Figure 6. Note, however, that the resulting consolidation still retains the same number of machines and connections to a switch port.

A second approach would be to use virtualization. The two servers can be converted into two virtual machines that are co-hosted on a physical server. The same number of physical NICs for the two VMs can be retained, as well as the wire-port connectivity to the switch. From a hardware perspective, the redundancy of network connectivity ensures that there is no single point of failure.

The administrator has several options when assigning

NICs to the VMs. An obvious choice would be to assign the physical NICs, one to each VM. However, this option loses the advantage of high availability. In fact, the NIC of a specific VM becomes the single point of failure for that VM's network. If that NIC fails, then all the VMs behind that failed NIC become unreachable. Furthermore, this setup restricts the scalability of the configuration to the limited number of physical NICs that can be installed on the bus as well as the number of ports on a switch.

A better approach would be to first create a link aggregation that bundles the physical NICs together. The aggregation is then virtualized into multiple VNICs and assigned to their respective VMs. Figure 7 shows this virtualized consolidation. In Figure 7, the VNICs are created based on the VLAN ID of their respective VMs. Thus, even after the transformation to a virtual environment is completed, traffic from the different VMs can still be differentiated on the wire.

Furthermore, every VM benefits from the HA of the networking connectivity because it has a redundant path to the network. An outage of one of the NICs or its port on the switch will result in a possibly slower overall network, however each VM is still reachable.

We show below the steps needed to create the link aggregation and then the VNICs to create the configuration of Figure 7.

```
# dladm create-aggr -l nxge0 -l nxge1 \
aggr0
# dladm create-vnic -l aggr0 -v 1 vnic1
```

Note that in this example, the single switch constitutes a single point of failure. Switch stacking or layer-3 multi-pathing can be combined with link aggregations to provide high availability across multiple switches, as described in Section 3.6.

5.2 Example 2 – Consolidating Multi-Tier Enterprise Networks

This example is a typical scenario for a cloud operator that offers hosting services for its enterprise clients. Each client tenant of the cloud operator's data center expects complete separation from the other tenants. This example demonstrates that all the three tiers (web server, App server, Database server and iSCSI storage) of the client data center as shown in Figure 8 can move to the cloud but remain isolated and separate from other virtualized data centers in the cloud.

The following steps show how to convert one of the client enterprise's Intranets. First create the etherstub for the Intranet and three VNICs on top of it.

```
# dladm create-etherstub stub1
# dladlm create-vnic -l stub1 VNIC_WS1
```

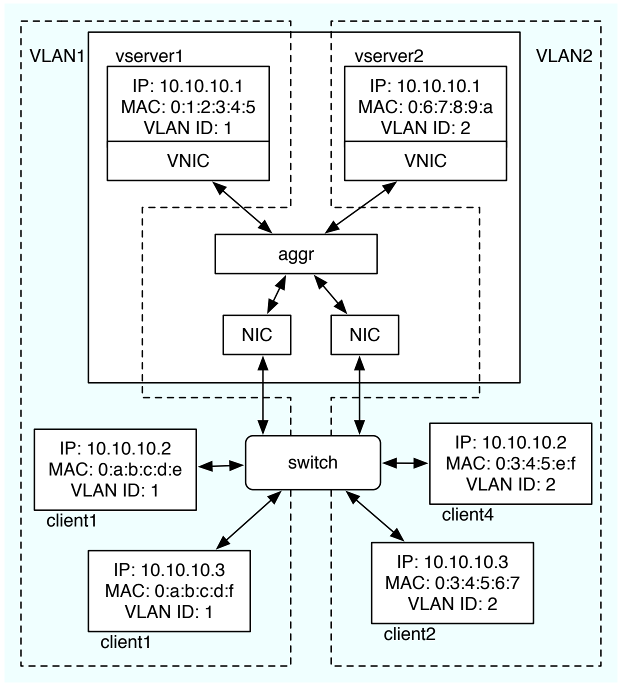

Figure 7: Example 1 – a vWire with two VLANs in a box

```
# dladlm create-vnic -l stub1 VNIC_AS1
# dladlm create-vnic -l stub1 VNIC_DB1
```

The VNICs can then be assigned to the zone `Webserver1` as described in Section 4.2.1. Similarly, assign `VNIC_AS1` and `VNIC_DB1` to `AppServer1` and `DBServer1`, respectively. Now connect the Database server to the back-end storage served by the iSCSI target: Create a VNIC on the back-end physical NIC:

```
# dladm create-vnic -l NIC2 VNIC_ST1
```

Assign `VNIC_ST1` to `DBserver1` as described in Section 4.2.1. Finally, connect the virtual enterprise subnet to the front-end edge router VNM by creating the `VNIC1` on the `Etherstub1` and assigning it to the Virtual Router VNM.

Figure 9 shows the resulting virtualized and consolidated Intranets for the two client enterprises. The physical servers have been converted into virtual appliances that are running in their respective zones. At the same time, the virtual network topology mimics the physical Intranets.

The two enterprises are competing for the CPU resources available on the virtualized server. Therefore, a remaining step is to define processor sets for each client, assign them to the zones, and bind the VNICs accordingly. Assume, for example, that `AppServer1` is assigned a processor set containing CPUs 1, 2, and 3. The VNIC can be bound to the CPUs assigned to `AppServer1` by issuing the following command:

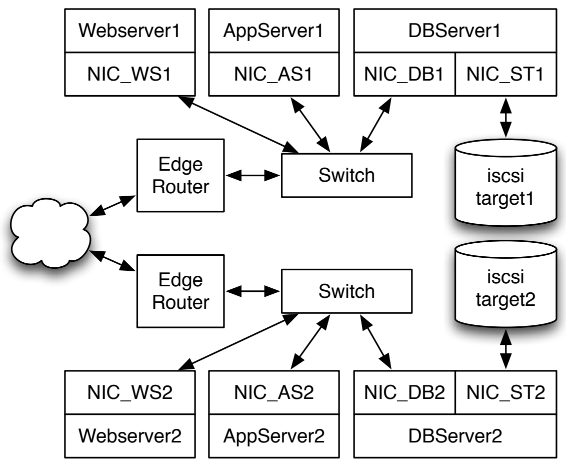

Figure 8: Example 2 – consolidating multi-tier enterprise networks, physical View

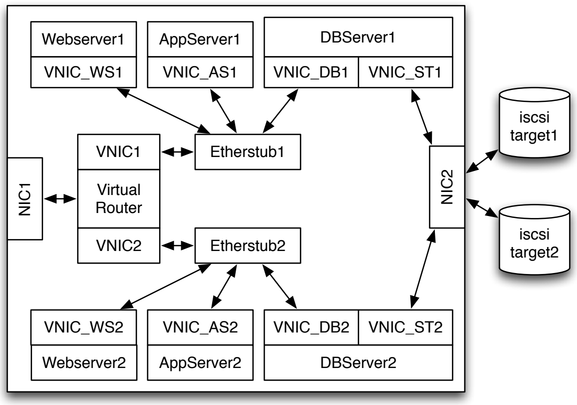

Figure 9: Example 2 – consolidating multi-tier enterprise networks, virtual View

```
# dladm set-linkprop cpus=1,2,3 VNIC_AS1
```

Future improvements will allow the data links to be automatically bound to the CPUs that are assigned to the zone, without requiring the administrator to manually bind the CPUs as shown above.

5.3 Example 3 – Try-Before-Deployment and Scale Out Scenario

In this example, we show how some of the observability and virtualization features of Crossbow can be employed to plan for scaling up the physical configurations as the need grows. The starting point is a small web server represented in Figure 10. As long as the amount of transactions coming from clients over the Internet is low, a single server is capable of handling the level of load required.

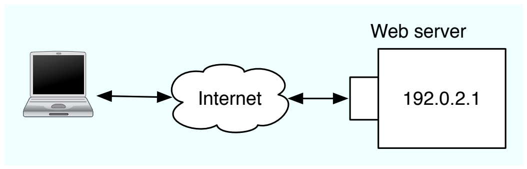

Figure 10: Example 3 – initial setting

In this scenario, the monitoring tools described in Section 4.3 can be used to log the usage history on the NIC to which the IP address 192.0.2.1 is associated:

```
# acctadm -e basic -f /var/log/net.log net
```

At this stage, only basic accounting for the networking interface is captured, and no flows are required. As the business picks up, the web server receives an increasing number of hits. A simple report to indicate the increased traffic activity can be obtained thus:

```
# dlstat -u -f /var/log/net.log
LINK     IBYTES  OBYTES  BANDWIDTH
e1000g     2.5M    0.1G 200.4 Mb/s
```

Anticipating further increase of traffic, the administrator can plan to horizontally scale the network up to multiple servers. However, before actually investing or committing any new physical resources to the network, it is desirable for the administrator to first understand how the new network configuration would actually behave while handling increased traffic. With Crossbow, the new distributed environment can be deployed and tuned in a virtual environment first.

In the give scenario, the web server is first virtualized into multiple virtual server instances running inside zones. Each instance can handle any of the URIs originally served. The virtual servers are connected to an in-box virtual switch through their respective VNICs. A load balancer and NAT appliance translates the IP addresses before forwarding the packet to the appropriate virtual server. An integrated load balancer [1] is expected to be available in OpenSolaris late 2009. Figure 11 shows the virtualized topology.

With the network usage history logging is still enabled, the amount of traffic on each link on the virtualized server can be monitored[2]:

```
# dlstat -u -f /var/log/net.log
LINK     IBYTES OBYTES  BANDWIDTH
```

[2]It is understood that most web servers also include logging of access statistics per URL. The authors' point here is to show how network infrastructure tools can be used for such accounting, whether the service being deployed included internal logging or not.

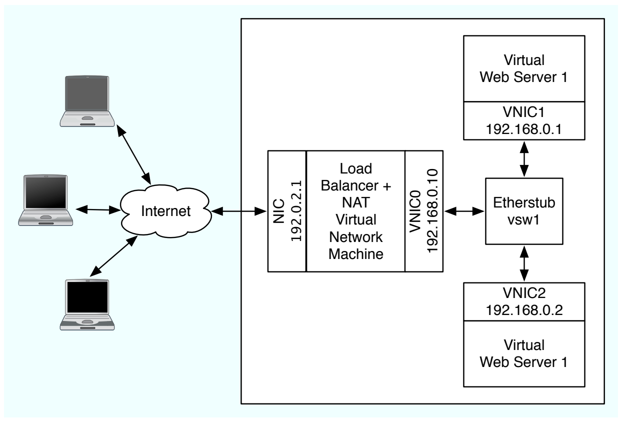

Figure 11: Example 3 – vWire for live workload analysis

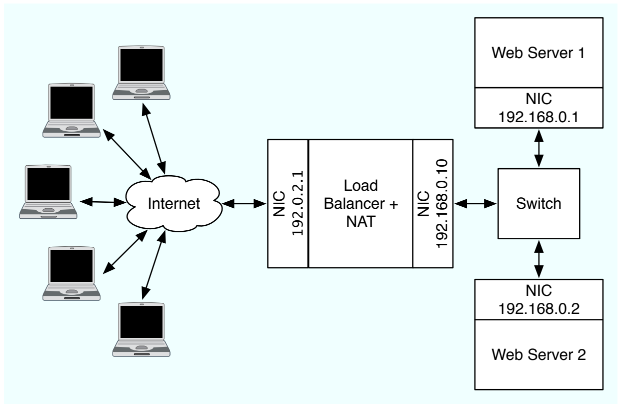

Figure 12: Example 3 – De-virtualizing for horizontal scaling

```
e1000g0    2.5M    0.1G   180.4 Mbps
vsw1       1.5M   52.7M   203.4 Mbps
vnic1      0.1M    3.0M    47.4 Mbps
vnic2      1.4M   49.8M   156.0 Mbps
```

This test run shows that the balance of traffic between the two virtual server appliances is imbalanced. The traffic through `vnic1` is only 23% of all traffic coming in the system, as opposed to the 77% being handled by the second virtual web server. The system administrator can then adjust the load balancer parameters to bring a more equitable distribution of the load.

When the load nears saturation levels for a single physical server to handle, the administrator can make an educated decision on the configuration of the new hardware. Note that the virtual web servers can be migrated to the new physical host with the exact same network configuration, without any need for IP renumbering. The final deployment is represented Figure 12.

It should be noted that more information can be derived from the usage history. The administrator could for example quantify the variation of load over time, and study the peaks of load, and the progression of the network usage, and extrapolate that progression to estimate the right time to start considering an upgrade.

6 Related Work

The Crossbow architecture provides mechanisms to achieve network virtualization within a host with ease of use and minimum performance penalty. The virtual NICs and flows leverage NIC hardware advancements such as classification and multiple receive and transmit rings to ensure the separation of virtualized packet streams without any processing overhead on the host. The virtual NICs and flows can be created over physical NICs, link aggregations, and etherstubs to provide private connectivity between virtual machines.

The idea of virtual switching has been implemented in other main stream virtualization technologies as well. Citrix System Xen [7] has a native Linux implementation where the physical NIC is owned by the hypervisor and virtual machines access the network by means of a front end driver that run in the guest domain and the back end driver that runs in the hypervisor. The hypervisor runs the physical NIC in promiscuous mode and uses a software based bridge implementation to provide all packets to the back-end drivers, which then select the packets that match their respective MAC addresses. There are mechanisms available to enforce bandwidth limiting and firewall rules on the traffic for virtual machines. However, these are typically separate subsystems, often very complex in implementation and administration, and can result in significant performance overheads [25]. VMware ESX based hypervisor has a proprietary implementation on a Linux variant but apparently suffers from some of the same issues [26] in terms of demultiplexing packets for various virtual machines and resource separation.

More recently, Cisco Systems announced a new virtualization offering under the Unified Computing System (UCS) [22] umbrella and based on the VMware EX hypervisor. The solution uses a specialized NIC along with a Nexus switch where packets from individual virtual machines are tagged to allow the switch to implement virtual ports and provide features similar to the Crossbow implementation. A centralized management solution in the form of a Virtual Supervisor module manages the physical and virtual components on the switch as well as hosts to provide easy management of resources and filtering policies. At the same time, the implementation is proprietary to Cisco software and hardware and VMware ESX hypervisor.

Some work is also occurring in the research community as part of the OpenFlow [11] consortium which helps in building a standard based programmable switch.

Such a switch would enable the Crossbow based hypervisor to program the switch with VLAN tags that are associated with customers and thus create more dynamic virtual networks where the switch can also provide separation, fairness, and security for the Crossbow vWire.

7 Conclusion and Future Work

The Crossbow virtualization and QoS components presented in this paper provide a unique mechanism to achieve network virtualization and consolidate multiple networks into one physical network. Assigning VLAN tags to VNICs and performing host based VLAN switching allow the creation of fully virtualized and isolated networks. Because the VNICs can be assigned link speeds, priorities, and dedicated NICs and CPU resources, a collection of virtual machines can span multiple physical machines and yet have deterministic performance characteristics. The configuration of VNICs and resource assignment is easy to configure and can be driven by external management tools with the provided APIs.

Apart from VNICs and virtual switches, multiple VNICs on different physical NICs can be assigned to OpenSolaris zones or virtual machines to create network components like routers, load balancers, firewalls, and so on. These virtual network machine along with VNICs and virtual switches can be combined together to create a fully virtualized network called vWire.

The Crossbow vWire offers a fully elastic, isolated, and dynamic virtualized network where virtual machines can migrate to other physical machines. The vWire extends with these VMs without needing any changes to the physical cabling or switches. Since the vWire uses VLAN tags and extended VLAN tags to provide isolation, it can work with any existing switch.

The various enterprise level features for failover and high availability such as link aggregation and IPMP, are designed in the architecture. Thus VNICs can be created over link aggregations and multiple VNICs on different attach points can be assigned to the same IPMP group. Care has been taken to ensure that a VNIC shows up as a separate interface on the MIB with the configured link speed as the interface speed. Existing network management tools can thus continue to work seamlessly in a virtualized environment.

The various examples in this paper show some of the possibilities where Crossbow can be used in an enterprise to decouple the application from the physical hardware and network to ensure easier deployment, management, and hardware upgrade. Because the vWire is a collection of rules and objects, it can be easily migrated from one physical network to another. This flexibility allows enterprises to migrate their network in full or in part to a public cloud when needed. The same concepts can be used by startups to create their data-center in a box in a public cloud. They can use Crossbow tools to analyze their usage and scale out to multiple machines seamlessly as business needs and traffic grow.

The core of the Crossbow architecture and all the features described in this paper have been implemented and integrated in OpenSolaris and available at http://opensolaris.org to any user.

Near term work focuses on enhancing the management tools to visualize and configure these vWires and virtual network machines. Crossbow has achieved a powerful level of control and observability over the networking resources inside a single system. One of the directions being pursued is to extend that kind of control beyond the boundaries of a single box, to encompass flows that span multiple subnets of physical and virtual machines. To that end, new wire protocols are being explored to convey some of the QoS requirements between nodes. We need to address both the data plane, and the control plane. Priority-based Flow Control (PFC) is the layer-2 mechanism defined by the IEEE and used for discriminating based on the VLAN tag's priority field on data packets. On the control plane, Generic Attribute Registration Protocol (GARP) and Multiple VLAN Registration Protocol (MVRP) are being considered for two reasons: The scalable administration of multiple interconnected nodes underscores the need for a hands off propagation of QoS information across the links. Secondly the network must be protected from the floods of unnecessary broadcasts from unused VLANs.

8 Author Biographies

Sunay Tripathi is a Distinguished Engineer at Sun Microsystem working on networking and network virtualization. He received a MS in Computer Science from Stanford University in 1997. His blog is at http://blogs.sun.com/sunay, and he can be reached at sunay.tripathi@sun.Com

Nicolas Droux is a Senior Staff Engineer and architect with the Solaris Core OS group at Sun Microsystems. Nicolas has led, designed, and implemented several kernel projects in the areas of High Performance Computing, I/O, security, virtualization, and networking. His blog is at http://blogs.sun.com/droux, and he can be reached at nicolas.droux@sun.com.

Kais Belgaied is a senior staff engineer and a technical leader at Sun Microsystems, Inc. His areas of interest include networking, virtualization, operating systems, cloud computing, and IT security. He is a voting member of the Platform Architecture Review Counsel with Sun Microsystems, and an active participant in multiple IETF

working groups. His blog is http://blogs.sun.com/kais, and he can be reached at kais.belgaied@sun.com.

Shrikrishna Khare is a Solaris Kernel Networking engineer at Sun Microsystems. He received a M.S. in Computer Science from North Carolina State University, USA in 2008. He can be reached at shrikrishna.khare@sun.com

References

[1] http://opensolaris.org/os/project/vnm/ilb (PSARC/2008/575).

[2] http://www.virtualbox.org.

[3] http://www.vmware.com/pdf/virtualization.pdf.

[4] M. Allman and W. S. V. Paxson. TCP Congestion Control. In *RFC 2581*, 1999.

[5] W. Almesberger. Linux Network Traffic Control.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. In *Technical Report No. UCB/EECS-2009-28*.

[7] 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.

[8] Carson. Limiting Bandwidth Usage on Xen Linux Setup.

[9] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield. QoS's Downfall: At The Bottom, or Not at All! In *RIPQOS'03: Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS*, 2003.

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

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Enabling Innovation in Campus Networking. 2008.

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

[13] L. M. S. C. of the IEEE Computer Society. IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks. In *IEEE Std 802.1Q-1998*, 1998.

[14] 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.

[15] T. H. Ptacek and T. N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. White paper, 1998.

[16] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a Denial of Service Attack on TCP. In *In Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 208–223. IEEE Computer Society Press, 1997.

[17] M. Seaman. A Multiple VLAN Registration Protocol (MVRP), 2003.

[18] 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.

[19] Sun Microsystems. Solaris 10 System Administration Guide: IP Services, 2008.

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

[21] Sun Microsystems, Inc. System Administration Guide: Solaris Containers-Resource Management and Solaris Zones, 2009.

[22] C. Systems. Unified Computing Systems.

[23] S. Tripathi, K. Belgaied, and N. Droux. Crossbow: Network Virtualization Resource Partitioning.

[24] S. Tripathi, N. Droux, T. Srinivasan, and K. Belgaied. Crossbow: From Hardware Virtualized NICs to Virtualized Networks. In *In Proceedings of the ACM SIGCOMM Workshop VISA'09*, 2009.

[25] 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*, 2009.

[26] 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.