CHAPTER

Server Virtualization and Networking

6

The computing power of a single server in cloud data centers has increased dramatically over the years to the point where certain workloads such as web hosting can significantly underutilize the processing power available. This has fueled the push toward microservers for certain workloads but has also spawned the widespread use of server virtualization. Server virtualization allows multiple virtual machines (VMs) to run as independent virtual servers on a physical server. The number of VMs running on a physical server depends on the workload requirements and server capabilities but can range in number from tens to hundreds.

Not all cloud data centers will utilize VMs and some hyper-scale data centers may implement proprietary methods not covered in this chapter. Software as a Service (SaaS) providers may or may not use VMs depending on how their software is written. Infrastructure as a Service (IaaS) providers let their customers make their own choices on how to use the leased hardware resource including the deployment of VMs. Platform as a Service (PaaS) providers will most likely use VMs in order to optimize their server resources in applications such as web hosting.

This chapter will provide an overview of how VMs work along with a key networking component called the virtual switch (vSwitch). This will be followed with a section on PCI Express (PCIe) which provides an alternative to the vSwitch. We will then describe Ethernet virtual bridging standards that were developed to better integrate VMs into the overall data center network. We will wrap up this chapter by discussing VM migration and the associated networking features that can improve this process.

VM OVERVIEW

Initially, host servers within the data center ran one operating system (OS) and, in many cases, one application program (app). As server CPUs became more powerful, these apps were consuming only a fraction of the CPU processing capability, but utilizing multiple apps on the same OS meant that an OS reboot would take down all of the apps running on the host. This led to the development of VMs. A VM is basically a guest OS running on a host server. For example, if you have ever loaded a program on your Apple® Mac computer that allows you to run Windows, you have set up something similar to a VM, and the app that you run to boot Windows on your Mac operates something like a VM hypervisor. The difference is that servers typically host many different VMs using a single hypervisor, each potentially running a different OS as shown in Figure 6.1.

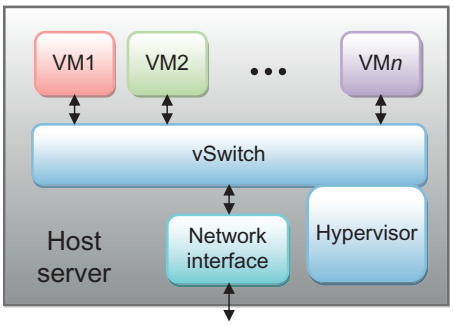

FIGURE 6.1

Logical diagram of a virtualized server.

Multiple VMs are logically connected to each other and to the network interface through a vSwitch, and a connection is made to the physical data center network through a network interface controller (NIC). A vSwitch is similar to the shared memory switch that we described in Chapter 3. A typical cloud data center server may have a 10Gb Ethernet NIC that connects to the server CPU through a PCI Express (PCIe) interface. Both the VMs and the vSwitch are set up and controlled by a hypervisor program that also runs on the server.

VMs allow more flexible allocation of server resources, especially in data centers that are running a large number of smaller applications such as web hosting. By using VMs, server utilization is increased, reducing the overall number of servers required in the data center. In addition, network interface bandwidth utilization is increased with multiple VMs sharing a single network connection. But this also drives the need for high-bandwidth Ethernet interfaces.

Originally, host servers were configured with one network connection per VM. As 10GbE NICs started to enter the market, it became more cost effective to share a single connection among multiple VMs. If you look at the data center network topologies that we described in Chapter 4, many people equate the vSwitch to a virtual top of rack (ToR) switch within the hypervisor, while the NIC provides the uplink ports for this virtual ToR that are shared among multiple servers. In this section, we will provide more information on the operation of the hypervisor while focusing on the network connections to the VMs.

Hypervisors

Hypervisors were first implemented on mainframe computers by IBM in the late 1960s. These were called bare metal, or type 1, hypervisors because they act much like an OS running on the physical hardware underneath. These hypervisors "hosted" what were called "guest" OSs which ran on top of the hypervisor. Another type of hypervisor is similar to the host program running on the Mac OS that was described earlier. This is called a hosted, or type 2, hypervisor that runs as an app on top of an

OS with one or more guest OSs running on top of the app. Other types of hypervisors include kernel-based virtual machines that run as kernel modules in Linux.

Virtualization technology did not find its way into the commodity server market until around 2005 when Unix and Linux systems started to become available. It wasn't until later in the decade that the use of VMs started to take off when Intel introduced x86 with hardware-based virtualization assistance. In addition, companies like VMware® and Microsoft started to introduce virtualization software that was deployed widely in data center installations. We will next provide more information about two of the leading commercial hypervisors in the market today. There are several other hypervisors available such as Kernel-based Virtual Machine (KVM) driven by the Open Virtualization Alliance that will not be covered in this chapter.

VMware

VMware was one of the first software companies to make a big splash in the virtualization market, and they are now one of the largest. They started with workstation virtualization software in the late 1990s and have since moved into the enterprise data center server virtualization market with vSphere™ (also called ESXi). This is a bare metal type 1 hypervisor which allows the VMs direct access to many of the server hardware resources. VMware also provides vCenter as a VM management tool. It provides a technology called vMotion™ that allows the movement of a VM from one server to another and from utilizing one storage device to another while the OS is running. It also provides high availability and load balancing features.

VMware also offers cloud management software called vCloud™ to manage logical pools of compute, storage, and networking resources. They recently acquired Nicira, a company developing software-defined networking solutions based on OpenFlow. These new developments show that VMware is moving toward solutions to help orchestrate and manage all of the key virtual components in large data center networks. We will provide more information on software-defined networking in Chapter 9.

Microsoft

Microsoft has developed a hypervisor called Hyper-V, which is designed for x86 64-bit servers. It is released as a stand-alone version and also with Windows Server OSs. Each server must contain one parent partition along with multiple child partitions. The parent partition creates and manages the child partitions and also handles all hardware access requests from the child partitions. Because of this, it is not exactly like a bare metal type 1 hypervisor but much more like a hosted type 2 hypervisor. With Windows Server 2012, Microsoft introduced live system migration features that allow the movement of a running VM and its storage from one Hyper-V host to another. This release also included features such as Hyper-V Extensible Virtual Switch and network virtualization technology.

VIRTUAL SWITCHING

Multiple VMs must share a single network physical port such as a network interface controller (NIC). One way to do this is to use PCIe single-root IO virtualization (SR-IOV) which will be discussed in the next section. The other way is for the hypervisor to implement a vSwitch in software. In this section, we will describe the vSwitch capabilities provided by Microsoft Hyper-V™ and VMware vSphere along with Open vSwitch (OVS) which is an open software initiative. At the end of this section, we will describe virtual machine device queues (VMDqs) which can be used to accelerate vSwitch operation.

vSphere Distributed Switch

VMware provides a virtual switching capability with their vSphere (ESXi) hypervisor. They call this a vSphere Distributed Switch (VDS) and it allows the extension of the vSwitch domain across multiple host servers as shown in Figure 6.2. Data physically moves between servers through the external network, but the control plane abstracts this movement to look like one large VDS spanning multiple servers. Each VDS can manage up to 500 hosts (servers). With VDS, the data does not move through a parent partition but logically connects directly to the network interface through local vNICs associated with each VM. The logical vNIC to physical NIC command and data translations are performed by the vSwitch.

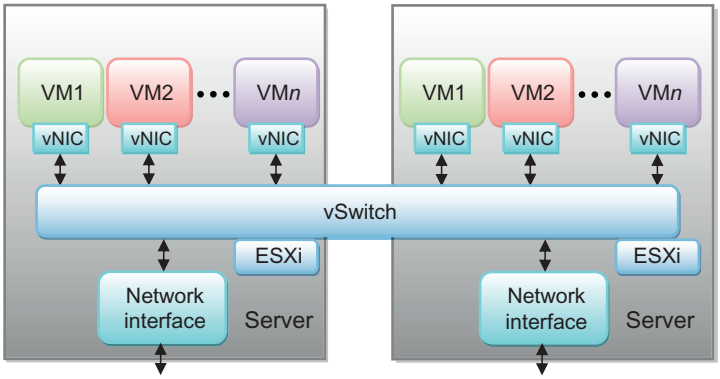

FIGURE 6.2

Logical diagram of a vSphere distributed switch.

VMware has added several advanced networking features to the latest VDS offering including the following:

- *Network traffic monitoring*: Enhanced network monitoring and troubleshooting capabilities
- *Virtual local area network (VLAN) isolation:* Provides network isolation in multitenant environments

- *Traffic shaping:* Provides maximum bandwidth limits to minimize traffic congestion
- *LACP support:* Support for link aggregation control protocol
- *Network vMotion:* Provides a simple way to monitor and troubleshoot when VMs are moved from host to host within a VDS

These features allow the VDS to provide offerings similar to what can be found in a physical network switch. But, like the Hyper-V switch described below, these advanced features require significant CPU processing resources as the number of VMs increase and as the network interface bandwidth increases, somewhat limiting the vSwitch scalability.

Hyper-V virtual switch

The Hyper-V hypervisor uses a parent partition to generate and manage multiple VMs referred to as child partitions. These child partitions communicate to the parent partition through a virtual bus called the VMbus as shown in Figure 6.3. All requests for access to hardware such as a network interface controller (NIC) are made through the parent partition. The parent partition also implements a virtual layer 2 switching function to provide switching between local VMs as well as arbitrating access to the physical network. It also provides a virtual NIC (vNIC) function for each VM that is connected to the VMbus. The vNIC looks like a physical NIC to the VM while the parent partition converts these local VM transactions into function calls to the physical NIC.

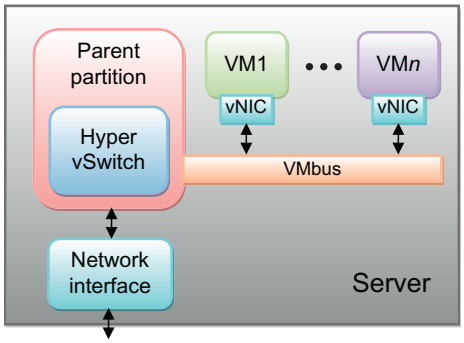

FIGURE 6.3

Logical diagram of a Hyper-V server.

The Hyper-V virtual switch also has three modes of operation; private, internal, and public. Private mode provides logical switching between VMs but no external access to the network. Internal mode is similar, but this mode includes virtual adapters (vNICs) within each VM. Public mode allows communication between VMs and also to the outside physical network through vNICs. With Windows server 2012, Microsoft added additional features in an attempt to provide vSwitch functionality similar to what can be found in physical data center network switches. These include:

- *Port ACL rules:* Access control list rules allow traffic classification and filtering.
- *Network traffic monitoring:* Allows network administrators to trace and monitor traffic in the virtual network
- *VLAN isolation:* Provides network isolation in multitenant environments
- *Minimum bandwidth guarantees and traffic shaping:* Provides minimum bandwidth guarantees along with maximum bandwidth limits
- *ECN marking:* Explicit congestion notification marking for congestion control
- *Network security features:* These include ARP spoofing protection, neighbor discovery spoofing, and DHCP guard protection.

Open vSwitch

OVS is an open source software industry initiative providing multilayer virtual switching capability while supporting standard management interfaces and protocols. It's the same vSwitch used in the OpenStack Neutron plugin and can also run on Microsoft Hyper-V and VMware's ESX hypervisors. Here is a feature list from the openvswitch.org web site:

- Visibility into inter-VM communication via NetFlow, sFlow(R), IPFIX, SPAN, RSPAN, and GRE-tunneled mirrors
- LACP (IEEE 802.1AX-2008)
- Standard 802.1Q VLAN model with trunking
- BFD and 802.1ag link monitoring
- STP (IEEE 802.1D-1998)
- Fine-grained QoS control
- Support for HFSC qdisc
- Per VM interface traffic policing
- NIC bonding with source-MAC load balancing, active backup, and L4 hashing
- OpenFlow protocol support (including many extensions for virtualization)
- IPv6 support
- Multiple tunneling protocols (GRE, VXLAN, IPsec, GRE, and VXLAN over IPsec)
- Remote configuration protocol with C and Python bindings
- Kernel and user-space forwarding engine options
- Multitable forwarding pipeline with flow-caching engine
- Forwarding layer abstraction to ease porting to new software and hardware platforms

Although these various vSwitch offerings provide features similar to physical network switches, in some cases they cannot meet the same performance levels. Because of this, hardware offloads such as VMDq and PCIe SR-IOV may be used in the NIC. These will be discussed in the next sections.

Virtual machine device queues

As the number of VMs increases within the server, the burden of managing all of this traffic within the vSwitch can tax CPU resources, which can reduce server and VM performance levels. To combat this problem, Intel developed VMDqs which are

implemented in many of their network controllers to both improve networking performance and decrease CPU utilization by offloading some of the vSwitch traffic management tasks.

Within the network interface controller (NIC), both transmit and receive queues are established for each VM hosted in the server as shown in Figure 6.4. When frames are received from the physical network, MAC address and VLAN tag information in the frame headers can be used to determine the correct destination VM. The vSwitch simply needs to forward the frame to the VM from the corresponding queue. When packets are transmitted from the VM, they are simply forwarded to the associated queue in the NIC. The NIC can then send frames out to the physical network based on various scheduling mechanisms such as round robin, weighted round robin, or strict priority based on service level agreements.

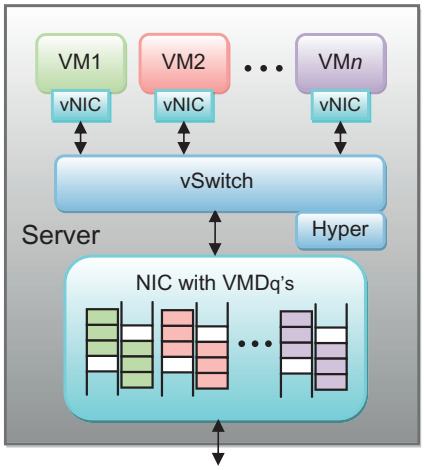

FIGURE 6.4

NIC with VMDq support.

VMs are assigned to cores within a multicore CPU on the server host. In some cases, a single core can burst data at rates over 5Gbps causing potential congestion in the NIC. The egress VMDqs can provide filtering and sorting capabilities to ensure proper egress link utilization under heavy traffic loads. On the ingress side, VMDqs can be used along with a scheduler to efficiently distribute the traffic load across the CPU cores in order to provide optimal performance. In some cases, VMDqs can be used in conjunction with a technology called Receive Side Scaling which is an industry standard mechanism supported by hypervisor vendors like VMware ad Microsoft to efficiently distribute traffic loads across multiple processor cores using a hash-based mechanism. Intel also offers a feature in their NICs called Flow Director which matches layer 2 and layer 3 header fields to determine which core to send particular flows to. This provides a unique association between the core and the client application.

PCI EXPRESS

PCI Express (PCIe) is an interface that you may be familiar with inside your PC. The PCIe interface standard has been used for years to connect peripherals such as graphics cards and network interface cards to the CPU chip set within the PC. This interface standard is also used for the connection between the CPU chip set and the network interface card within the server. Although it has been around for a while, new industry standards have evolved under the PCIe umbrella that provide enhanced functionality and performance when supporting multiple VMs within a server. This section will provide some background on PCIe and describe both the single root IO virtulization (SR-IOV) and multiroot IO virtualization (MR-IOV) standards.

Background

Early computers and workstations used various parallel interfaces such as the ISA or VESA local bus to connect peripheral devices to CPU subsystems. In order to provide a single industry standard interface for its CPU chipsets, Intel started working on what became the Peripheral Component Interconnect (PCI) standard in the Intel Architecture Development Lab around 1990. This work propelled the computer industry to band together to form the nonprofit Peripheral Component Interconnect Special Interest Group (PCI-SIG) in 1992 in order to standardize the PCI interface. Today the PCI-SIG has over 800 corporate members and this standard is widely used in the industry.

The original PCI interface is a 32-bit wide parallel bus operating at a 33MHz clock rate and it quickly replaced the ISA and VESA local bus in most applications. The PCI standard includes bus timing requirements, protocol specifications, electrical specifications, and the physical dimensions of the PCI edge card connector. In some applications, a connector is not used and the bus connects to another device located on the same circuit board. Also, in some cases, the CPU chipset supports only a single PCI interface, so a PCI bridge device is used to expand this single connection to multiple PCI interfaces using a bus-like architecture within the bridge. The host OS enumerates each PCI component on the bus and assigns an address range to each device which can be used by the host to configure, monitor, and communicate with the peripheral. Only one host or root device can be used in this system.

The PCI standard was quickly accepted by the industry, and the PCI-SIG set to work to increase PCI performance levels. There are two ways to increase the performance of a parallel bus; increase the bus bandwidth or increase the bus width. For PCI, this was done in stages as follows:

- 133MBps—32-bits wide at 33MHz (original standard)
- 266MBps—64-bits wide at 33MHz
- 266MBps—32-bits wide at 66MHz
- 533MBps—64-bits wide at 66MHz

The PCI-SIG later introduced the PCI-X standard that not only increased the performance but also improved the fault tolerance of PCI by allowing defective cards to be reset or disabled. The commonly used PCI-X bus standards are as follows.

- 533MBps—64-bits wide at 66MHz (found on older servers)
- 1066MBps—64-bits wide at 133MHz (common implementation)

There are also 64-bit wide 266MHz and 533MHz standards for PCI-X, but these have been rarely implemented and were quickly replaced by the PCIe standard.

As we discussed in Chapter 3, wide parallel busses are difficult to implement in circuit board designs as the bus width and bus frequency increase. These wide busses also consume a large number of pins, taking up area and increasing the cost of integrated circuits. Because of these factors, the PCI-SIG decided to make a significant change by introducing the PCI Express (PCIe) standard in 2004. PCIe can use multiple data lanes, but instead of using single-ended signals between devices, it uses higher performance serializer/deserializer (SerDes) technology with differential signaling. These SerDes can increase performance while easing board layout and reducing pin count. As of this writing, three generations of PCIe standards have been released by the PCI-SIG, each with increased SerDes transfer rates. In addition, various PCIe transfer rates can be supported by using different numbers of SerDes (lanes) as shown in the table for each PCIe generation.

Gen	1-lane	2-lane	4-lane	8-lane	16-lane	32-lane
1	250MBps	500MBps	1GBps	2GBps	4GBps	8GBps
2	500MBps	1GBps	2GBps	4GBps	8GBps	16GBps
3	985MBps	1.97GBps	3.94GBps	7.88GBps	15.8GBps	31.5GBps

For Ethernet network interface controllers, common PCIe interface standards are 4-lane gen2 for 10GbE data traffic (1.25GBps) and 8-lane gen3 for 40GbE data traffic (5GBps). The use of SerDes technology not only increases the interface performance while using fewer pins, it also allows more flexible interconnect topologies with multiple devices connected to the host through a PCIe switch, which logically behaves like a PCI bus. An example PCIe peripheral configuration is shown in Figure 6.5.

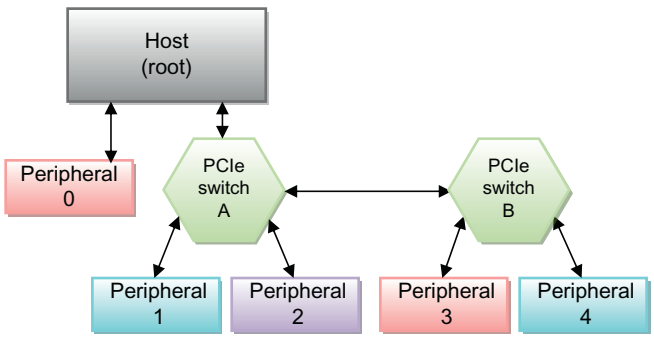

FIGURE 6.5

Example of PCIe peripheral connections.

As you can see, the PCIe switch hierarchy looks like a tree structure with the host CPU referred to as the root of the tree. There are actually three different virtual PCI busses in this example. The host contains an internal virtual bus allowing it to access both peripheral 0 and PCIe switch A. PCIe switch A contains a virtual bus with connections to two different peripherals as well as PCIe switch B. This switch contains the third virtual bus with connections to two additional peripherals. Each peripheral contains at least one physical function (PF) which is connected to the host. We will not dive deeply here into the details of the PCIe protocol; instead, the key take away from this discussion is that any given peripheral can only be connected through the tree to one root (host). This will become important in the next sections.

Single-root IO virtualization

We mentioned above that there can only be a single root host connected to any given peripheral such as a network interface controller. But a single host may be running a hypervisor with multiple VMs that would like to share this network interface. We also mentioned earlier that hypervisors can implement vNICs for each VM, but as network interface bandwidths increase, the vNICs start to tax CPU performance. To provide an alternative, the PCI-SIG issued an extension to the PCIe specification called single-root input output virtualization (SR-IOV). This spec moves the vNIC functionality into the network interface card as shown in Figure 6.6, improving overall performance.

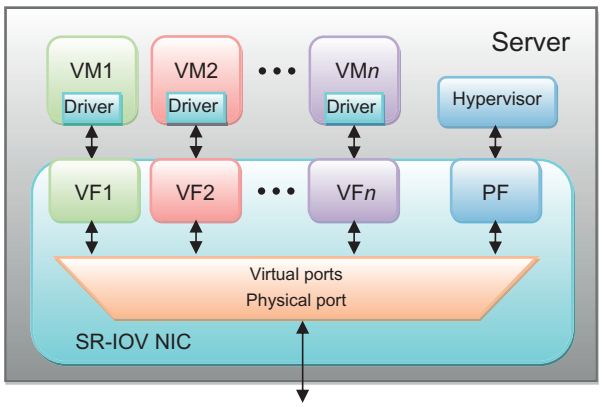

FIGURE 6.6

Conceptual drawing of SR-IOV components.

Instead of having one physical function (PF) like traditional PCI NICs, the SR-IOV NIC contains one PF along with multiple virtual functions (VFs). Each VF has all the features and functionality of a NIC and also has its own PCI configuration

space. This allows each VM that is using an SR-IOV compatible NIC driver to function like it's connected to a dedicated NIC. The SR-IOV PF takes care of allocating physical port bandwidth to the virtual ports associated with each VF. For this to work, the hypervisor must also support SR-IOV. Both Microsoft and VMware support this capability as long as specified NICs from certain vendors are used.

The hypervisor communicates to the PF which exposes to the hypervisor the SR-IOV extended capabilities in the PCI configuration space. This allows the hypervisor to configure and manage the SR-IOV NIC including the association of VMs with VFs. SR-IOV capable NICs from different vendors can support different numbers of VFs, so the hypervisor must read this information from the PF configuration space before assigning VMs to VFs. You can think of SR-IOV as a hardware offload of the vSwitch network connections that are implemented in the NIC to reduce CPU utilization and improve performance.

A vSwitch can be thought of as a shared memory switch in which only the memory pointers need to be passed between VMs sharing the same switch. When using SR-IOV, traffic between two VMs within the same server must pass through the NIC PCIe interface before returning back from the NIC. With today's servers connected to a 10GbE NIC, the PCIe bandwidth is limited to around 14Gbps while a vSwitch can maintain transfer rates up to 30Gbps because it is simply passing memory pointers. This performance advantage is becoming more important as the amount of east-west traffic increases within the host servers. Although the vSwitch uses more CPU resources than required with SR-IOV, high-performance multicore processors can easily handle this extra burden. Because of this, along with other factors such as SR-IOV management overhead and resource coordination, SR-IOV is typically used in specific applications where vSwitch CPU overhead and/or latency are an issue.

Multiroot IO virtualization

To reduce cost and improve system flexibility, it would be ideal if multiple CPU subsystems within a chassis could share IO resources such as NICs for networking, host bus adapters (HBAs) for storage, and host channel adapters (HCAs) for computer clustering applications. This was the idea behind the MR-IOV standard developed by the PCI-SIG under the PCIe umbrella.

Figure 6.7 shows an example application for MR-IOV where multiple servers are sharing different IO adapter devices. Each adapter must have special features including support for multiple virtual PCIe interfaces and VFs in addition to a physical PCIe interface and PFs. The MR-IOV PCIe switch must have special functionality to align server transactions to the correct virtual adapter interface. This configuration can be done through a dedicated control plane CPU connected to the switch or through one of the attached servers. Similar to SR-IOV, the trick is to make sure each server driver thinks it owns the resources in each adapter to which it is logically connected.

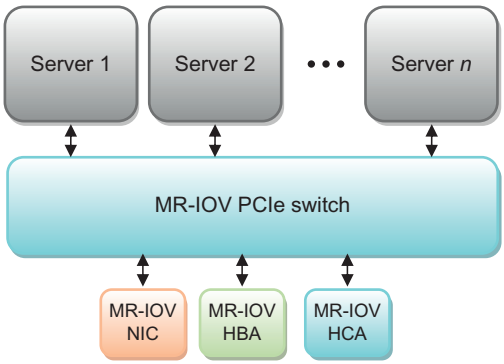

FIGURE 6.7

Example MR-IOV implementation.

MR-IOV has not seen much use in cloud data centers because powerful multicore CPUs that can host multiple VMs need the full bandwidth of the network adapter, so sharing the adapter among multiple CPUs does not make sense. In other applications, it only makes sense where multiple CPUs need to share relatively expensive adapters, which limits the available market. Several companies, such as NextIO and Virtensys, developed IO virtualization systems built on the MR-IOV specification, but both have now ceased operations.

EDGE VIRTUAL BRIDGING

Even though hypervisor vendors have added a lot of networking features recently to their vSwitch offerings, they still lack some of the advanced features that are available in the attached top of rack switch. Because of this, traffic between VMs within the same server may not be subject to the same filtering, security, monitoring, and forwarding capability as traffic that is sent through the top of rack switch. To improve this situation, the industry has been working on some new specifications that are collectively known as edge virtual bridging (EVB). This section will describe two key EVB standards that have been promoted in the industry; virtual Ethernet port aggregator (VEPA) and virtual network tag (VN-Tag).

VEPA

VEPA is a method for providing consistent treatment of all network traffic including VM traffic. This was originally proposed by HP® and has since been used by the IEEE as a basis for the 802.1Qbg EVB standard. With VEPA, all the Ethernet frames generated by a VM are routed through the attached network switch in order to provide consistent treatment of all network traffic as shown in Figure 6.8.

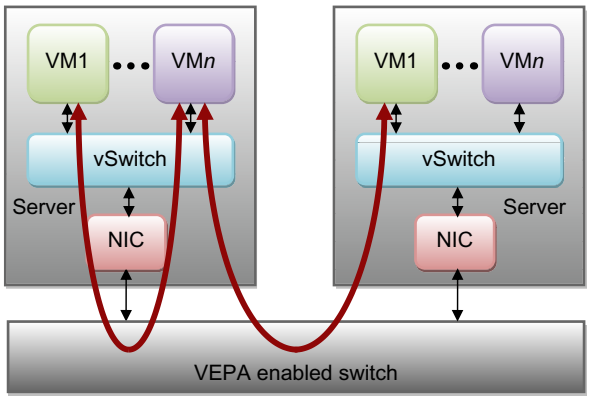

FIGURE 6.8

Forwarding VM traffic using VEPA.

There are two VEPA modes of operation. In standard mode, the hypervisor software must be configured so that all traffic generated from a VM is forwarded to the external switch. For the external switch, this does not require any changes for traffic that is intended for another physical server, but if the traffic is intended for a VM within the same server, the switch must forward the packet back through the same port that it was received on. This is sometimes known as reflective relay, or a hair-pin turn. Because the spanning tree protocol does not allow a frame to be forwarded in this way, the switch may require a firmware upgrade in order to support this. The advantage of this approach is that all traffic is treated in a consistent manner by the attached switch. The disadvantage is that performance may be impacted as traffic that would normally stay within the vSwitch must now be forwarded in both directions through the network interface controller, increasing the bandwidth utilization of this device. The switch must also support a Virtual Service Interface, which makes it appear that the VM is directly connected to the external bridge.

VEPA also supports an optional channelized mode of operation in which all traffic has an extra VLAN tag added to the frame and the VLAN ID represents a channel number. This type of tunneling is also known as Q-in-Q encapsulation. The channel number can be used to isolate traffic from different VMs or different vSwitches within a single physical network connection. The attached switch can also use this channel identifier to provide different service levels to different VMs.

VN-Tag

VN-Tag is also a method for providing consistent treatment of all network traffic including VM traffic. This was originally proposed by Cisco and has since been used by the IEEE as a basis for the 802.1qbh Bridge Port Extension standard. Cisco has been a proponent of the fabric extender concept (using their Nexus 2000 product) that

we described in Chapter 4. This approach effectively distributes the line cards of an end-of-row switch to each rack as shown in Figure 4.5.

The controlling bridge configures the port on the fabric extenders as if they were a physical interface on the controlling bridge. To do this, the controlling bridge creates a logical interface for each port in the fabric extender and assigns a tag value. The fabric extender uses this value to add tags to all traffic moving through these ports. Figure 6.9 shows how this concept can be extended to the VMs within a server.

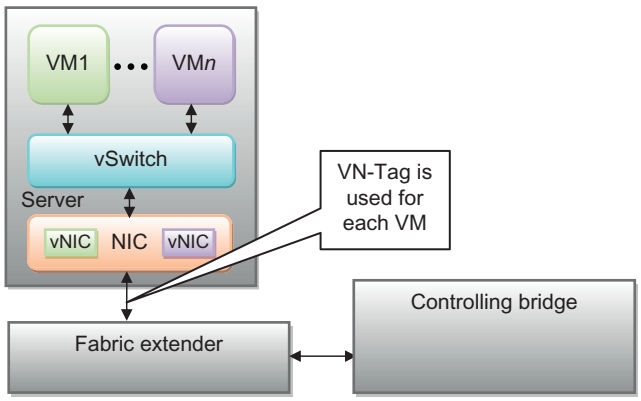

FIGURE 6.9

VN-Tagging.

The use of VN-Tags expands the reach of the controlling bridge into the NIC. A single physical NIC can be configured to contain multiple vNICs each with its own assigned VN-Tag. So now the controlling bridge has extended its reach not just to the external ports of the fabric extender but also into the ports of the vNICs within the server. While the VEPA standard works by extending traffic flows out to the attached switch, VN-Tag works by extending the functionality of the controlling bridge into the server.

Industry adoption

The goal of both VEPA and VN-Tag is to provide more consistent networking policies down to the vSwitch level. In the case of VEPA, hypervisor support is required which in not yet widely available due to the fact that many vSwitch implementations have recently added capabilities traditionally found in the physical networking switches. In the case of VN-Tag, many people still consider it to be a proprietary approach with special hardware requirements such as specialized NICs which has limited it's acceptance in the market. In any case, hypervisor vendors are extending advanced networking capabilities in their vSwitch offerings and advanced network virtualization and tunneling standards are emerging including VXLAN and NVGRE which provide new ways of moving data between VMs which we will describe in the next chapter.

VM MIGRATION

In some cases, a running VM must be moved from one host to another due to failure, maintenance, a performance upgrade, or for network optimization. This is known as VM migration, live migration, or also by brand names such as vMotion from VMware. Most customer applications cannot tolerate downtime while this migration happens. The migration is called "seamless" when the downtime is not noticeable by the end user. There are several ways that the network can impact this migration. First of all, moving all of the current state information from one host to another requires low latency networking with lossless operation. In addition, the network addressing and forwarding must be updated quickly and seamlessly once the migration is complete. In this section, we will discuss VM migration and how the data center network may affect this operation.

Memory migration

Before a running VM can be migrated, all of its memory, storage, and state information must be moved to the new VM, which is configured but not yet running on the new host. The new VM must be configured with the same OS and applications that were running on the old VM. Then all of the storage data, local memory data, and CPU state information must be moved over before the new VM can be started and the old VM shut down. Here are some steps required for memory migration.

Move storage

The first step is to duplicate storage from a hard drive or solid-state drive on the old host to a drive on the new host over the network. This is known as storage migration. This needs to be completed quickly so that only a limited number of storage sectors become stale during the move and the running VM can keep a log of storage sectors that need to be refreshed after the move. To complete this quickly, low latency networking is key. The separation of storage and data traffic using techniques like DCB that we discussed in the last chapter will help improve storage migration performance. In some cases, the running VM may be using pooled storage or storage that is not local to the host. This actually simplifies storage migration as the new VM needs to only learn the address information of the storage location within the network. More information on network migration will be provided below.

Move memory

For memory migration, the hypervisor copies pages of memory from local DRAM on the old VM to the new VM over the network. Some pages may become stale during this operation due to changes, so the copy process may be repeated for some pages during the transition. Again, speed is critical to this operation so low latency networking is important and technologies that we described in the last chapter such as remote direct memory access (RDMA) can help improve the performance during this process.

Move CPU state

Once the storage and memory are moved over, the old VM operation can be suspended while the execution state and register state of the CPU is moved over to the new CPU. Next, the new VM is started up. The downtime is an important factor and it's typically not noticeable if it's less than a second. After the new VM is brought to life, there may still be some clean-up work to do in order to make sure memory and storage are consistent between the old VM and the new VM.

Network migration

Another key factor in VM migration is updating the network to reflect the new location of the VM in the data center. In a traditional layer 3 cloud data center, forwarding of frames to the VM relies on forwarding tables located in each switch. For example, when a frame arrives in a given switch, the switch may inspect the IP destination address to determine where to forward the frame. If that IP address has moved to another location in the network, all of these tables need to be updated in the relevant network switches.

Today, many data center networks are set up to tunnel frames from within the vSwitch or top of rack switch at the network edge. We will discuss this in more detail in the next chapter on network virtualization. The point here is that if this type of tunneling is used, only the tables at the network edge need to be updated. In some cases, a given public cloud tenant may use only a few VMs in the data center so only the edge switches associated with these VMs will need to be updated. In any case, these updates need to be well timed with the VM migration and certain frames in flight may be dropped and retransmitted until the new VM is up and running.

Figure 6.10 shows an example of how the data center network may be affected by VM migration. Let's assume a data center tenant is running applications on two VMs that are located on different data center racks and they are accessing storage on a third rack as shown on the left side of Figure 6.10. This is inefficient and consumes

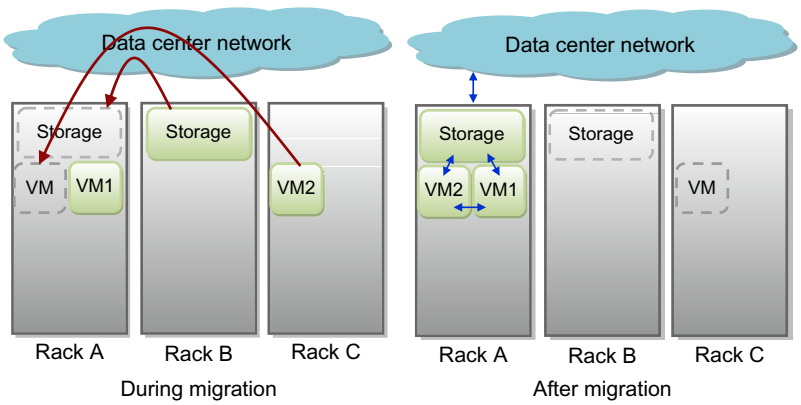

FIGURE 6.10

Network utilization during and after VM migration.

unnecessary bandwidth in the data center network. The ideal situation is shown on the right side of Figure 6.10 where the two VMs reside within the same host server within the rack and the shared storage is also within the same rack. Now the VM to VM traffic runs on the local vSwitch within the server and storage to VM traffic runs on the network within the rack. Only client traffic runs out of the rack, reducing the load within the data center network.

Once the data center server administrator notices this potential optimization, a VM migration process may be initiated as shown in the left side of Figure 6.10. This process of quickly moving storage and server state information across the data center network requires high network bandwidth and may add temporary hot spots to the network. The network administrator must allocate the necessary bandwidth using mechanisms such as data center bridging which we described in the last chapter in order to make sure this migration is completed with minimal downtime. Another factor to consider is that during the migration process, the high-bandwidth NIC operation on the servers hosting these VMs may consume CPU resources, slowing down other applications running on those CPUs.

As you can see from the discussion above, there is critical timing required from both the server administrators and the network administrators to make sure VM migration goes smoothly. A new approach to this is emerging in the industry. It is called software-defined networking and, in conjunction with VM migration software and a central orchestration layer, it can help automate this process and eliminate the human errors that may cause problems. We will provide more information on software-defined networking in Chapter 9.

Vendor solutions

The discussion above was a simplified view of what happens during VM migration. In reality, there are many other moving parts and detailed actions that need to take place. This is where software vendors can really differentiate themselves with VM migration solutions that minimize downtime and maximize ease of use. For example, Microsoft provides live migration capabilities in their Hyper-V product offering which has features to eliminate downtime or the need for shared storage. IBM and Oracle® are two leading server OEMs that also provide tools for live migration. Many of these tools can take advantage of RDMA technology such as iWARP to provide low latency transactions during VM migration in order to minimize downtime.

VMware provides vMotion as well as Storage vMotion as a way to provide live migration between hosts without disruption or the need for shared storage between the hosts. vMotion uses a bit-map to keep track of ongoing memory transactions in order to ensure transaction integrity after the move. Recently, VMware purchased an SDN company called Nicira and now provides their virtualized networking capability under the NSX branded products. It is expected that VMware will expand their vMotion capabilities to include several new network virtualization features that will help automate the VM migration process. We will provide more information on SDN in Chapter 9.

REVIEW

In this chapter, we described VMs and how hypervisors from some leading software vendors establish virtual switches within the host for local connectivity. Next we provided an overview of PCIe and described how SR-IOV can provide an alternative to a vSwitch in the host. We then provided some details on how the physical and virtual switches interact and how the industry is establishing EVB standards in order to unify network functionality down into the servers. Finally, we discussed VM migration and the requirements this poses on the networks. In the next chapter we will complete this story with network virtualization which can allow multiple tenants to have independent virtual networks within a large cloud data center network.