Intel® VMDq Technology

NOTES ON SOFTWARE DESIGN SUPPORT FOR INTEL VMDQ TECHNOLOGY

Network Division (ND)

March 2008
Revision 1.2

Legal

Contents

Revisions

Date	Revision	Description
February 2008	1.0	Initial release.
March 2008	1.1	Corrected minor error.
March 2008	1.2	Updated list of controllers supporting VMDq.

1 Introduction

Virtual machines (VMs) have been used for decades in mainframe computers to allow multiple copies of different operating systems to run concurrently on a single hardware platform. In the past, Intel® Architecture (IA) systems did not have all the processor and platform features that would allow the mainframe-style virtualization and software solutions that were developed in the 1990s. Intel has since introduced technologies to provide better performance in a virtualized environment.

This document provides an overview of the Intel VMDq technology available in many Intel® PCIe* v2.0 (2.5GT/s or 5GT/s) Ethernet controllers and discusses how a Virtual Machine Monitor (VMM) (also called a Hypervisor) might utilize this technology to improve network performance in a virtualized environment. The document focuses on the 1st generation of Intel VMDq technology. It is designed to be a companion document to a specific product's software developer's guide; refer to Section 4 for a list of applicable Intel® Ethernet Controllers.

Readers of this document should have an understanding of Ethernet, IP, TCP/IP, and VLAN. They should also have a basic understanding of operating system internals and application usage models for local area networks, as well as a basic understanding of server virtualization. See Figure 1.

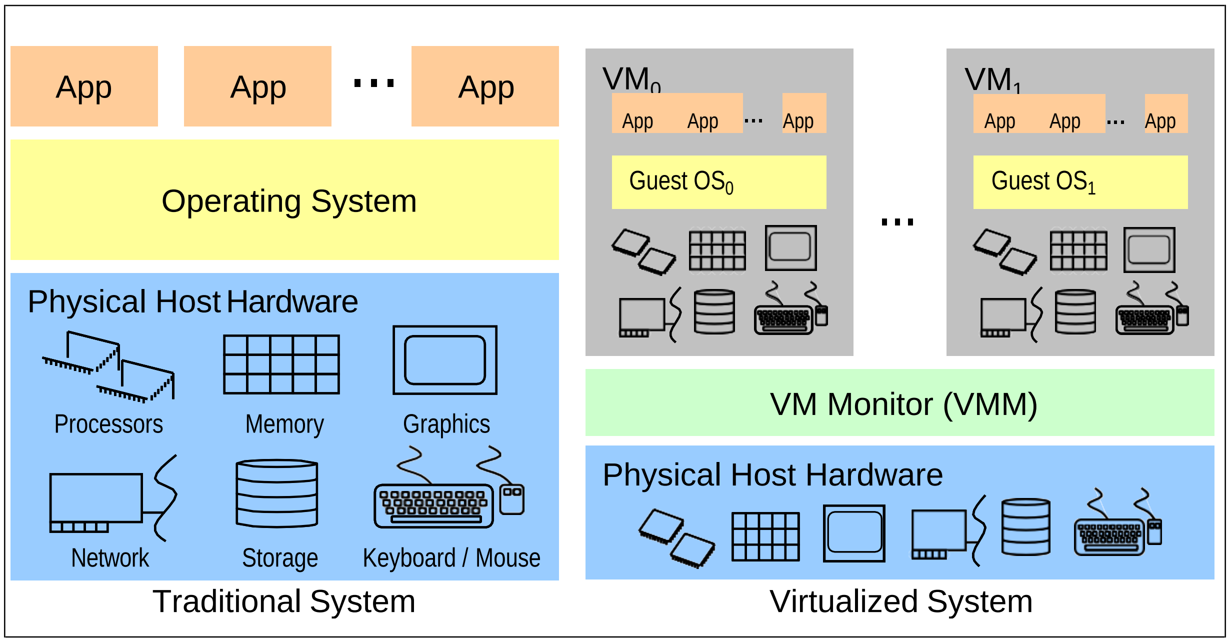

Figure 1. Example Systems

1.1 Terminology

The following terms are used in this document.

EEDC	Intel's Enhanced Ethernet for Data Center architecture
I/O Virtualization	The capability for a single I/O unit to be used by more than one System Image.
IOV	I/O Virtualization
iSCSI	Internet SCSI
Guest Physical Address (GPA)	Guest Physical Address is the view of physical memory from software running in a partition.
PCIe	PCIe v2.0 (2.5GT/s or 5GT/s)
PCI I/O Virtualization	The capability for a single physical PCI I/O adapter to be used by more than one System Image.
Physical Address	The address used by the system memory controller to access system memory.
RDMA	Remote Direct Memory Access
SW	Software – driver code appropriate for the respective, supported Operating Systems
Virtualization	The construction of a set of VMs by the division of a physical system's processors, memory, I/O, and storage. Each such set of resources operates independently with its own System Image and applications. Each VM communicates with the other VMs as if the other VMs were in a separate machine.
Virtual Machine (VM)	The collection of physical or virtualized resources necessary to run a single System Image instance. The virtual resources typically consist of processors, memory, I/O and storage
Virtual Machine Monitor (VMM)	A software layer that creates an abstraction of a virtual host system; also called a hypervisor.
VMDq	Virtual Machine Device Queue – A LAN Device that provides queues for SW based NIC sharing.
VT-d	Intel® Virtualization Technology for Directed I/O - VT-d provides VMM software with the capabilities to Assign I/O devices across VMs, to support DMA remapping and additional reliability.
VT-x	Intel® Virtualization Technology for IA-32 Processors - VT-x constitutes a set of virtual-machine extensions that support virtualization of processor hardware for multiple software environments by using VMs

1.2 Network I/O Bottleneck

Deploying virtualized environments on powerful platforms is a growing practice among IT departments in order to consolidate server workloads and reduce data center footprints. The practice, however, can have a significant impact on system and application performance as workload efficiency increasingly depends on network I/O.

As IT managers add greater processing power and reduce infrastructure footprint, this consolidation does not necessarily mean more efficient network throughput in the virtual environment. A balance between system performance and networking capabilities is required to achieve optimal application services from consolidation.

In virtual environments today, the VMM manages network I/O activities. With more VMs and increased traffic through the platform, VMMs require more CPU cycles to sort data packets and route them to the correct VM (see Figure 2). This requirement can reduce CPU capacity available for applications.

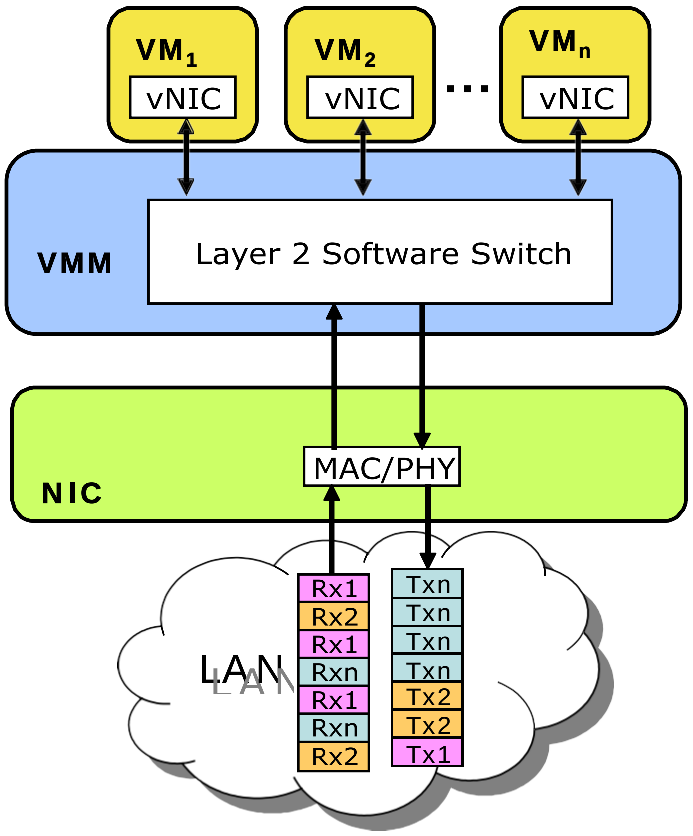

Figure 2. Software Based Packet Switching

1.3 Virtual Machine Device Queues

Virtual Machine Device Queues (VMDq) are part of the Intel® Virtualization Technologies. VMDq is a silicon-level technology that offloads some of the network I/O management burden from the VMM (see Figure 3).

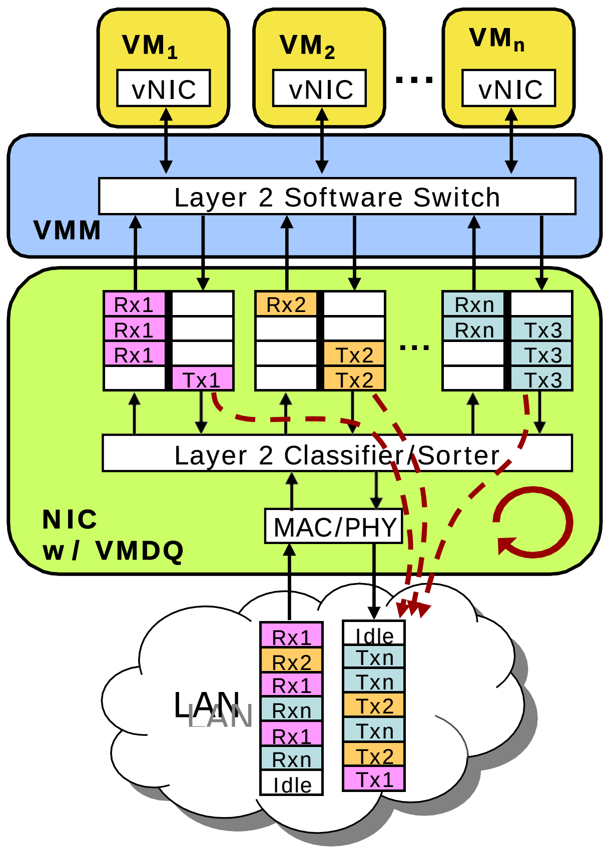

Figure 3. VMDq Packet Sorting

When packets arrive at the VMDq Enabled and configured network adapter, a Layer 2 classifier/sorter in the network controller sorts and determines the correct destination queue using MAC addresses and VLAN tags. The sorter then places the packet in the receive queue assigned to that VM. The VMM's 'switch' routes the packets to the respective VM; this keeps the VMM from having to perform the heavy lifting work of sorting the data itself.

As packets are transmitted from the VMs to adapters, the VMM layer places the transmit data packets in their respective queues. To prevent head-of-line blocking and ensure each queue is fairly serviced, the network controller transmits queued packets to the wire in a round-robin fashion. This guarantees some measure of Quality of Service (QoS) to VMs using queues.

1.4 VMDq Performance Impacts

VMDq technology provides performance gains by increasing network traffic throughput and reducing CPU utilization in a virtualized environment.

1.4.1 VMDq Bridge Overhead Reduction

Analysis of the cost of sharing a physical adapter in an I/O container model indicates that bridging overhead is second only to the overhead of moving the data from an I/O container to a VM. Software virtual switches typically are copied from non-virtualized solutions which feature functionality over performance.

The VMDq pre-sorting offload can change “virtual switch total overhead per data flow” from “virtual switch per packet overhead times number of receive packets” to “virtual switch per packet overhead divide by number of packets in group receive.”

1.4.2 VMDq I/O Container Move Overhead Reduction

The primary overhead in I/O sharing with an I/O container is ‘moving’ packets from the I/O container to the VM. The ability of VMDq to ‘direct DMA’ reduces this overhead but does not fully eliminate move overhead.

1.4.3 VMDq Transmit Overhead Reduction

Transmit fairness mechanism is a critical factor in a shared LAN adapter. A key issue is ‘head of the line blocking’ in which a VM transmit must wait behind transmits generated by other VMs. Assuming uniform distribution of transmit traffic between multiple VMs, head of line blocking can reduce an individual VM network throughput by as much as 59% of theoretical maximum.

2 Intel VMDq and Virtualization

2.1 Hardware Features

VMDq hardware features are:

- Layer 2 Filters for sorting packets based on MAC Address
- Layer 2 Filters for sorting packets based on VLAN tags[1]
- Typical implementations support eight or more RX queues
- Typical implementations support eight or more TX queues
- Supports single transmit descriptor queue fetch arbitration mode of round robin
- Filters can be applied while adapter is in promiscuous mode
- Default queue mechanism for non-matching packets
- MSI-X Interrupt per queue
- Functions with or without VT-d

2.2 VMDq Level 1 & Level 2

Software for VMDq can be implemented at two levels. **VMDq Level 1** adds support to reduce the overhead of Virtual Bridges/Switch and provides Transmit 'Fairness'. **VMDq Level 2** adds support to reduce the overhead of "moving" packets from an I/O Server/Container to the VM.

- VMDq Level 1 support:
  - Able to configure multiple HW filters/queues
  - Supports a method to indicate a group of receive packet match a configured receive filter
  - Support for transmit fairness
- VMDq Level 2 support:
  - Supports Level 1 features
  - Support for targeted receive to reduce move overhead

2.3 Software Components

Figure 4 and Figure 5 describe how major software components typically interact with VMDq devices. The sections that follow describe components in the diagrams.

[1] VMDq has certain limitations on sorting packets based on VLAN tags. VMDq only sorts packets based on MAC address or VLAN tags. VMDq can not differentiate between two packets with the same MAC address and different VLAN tags.

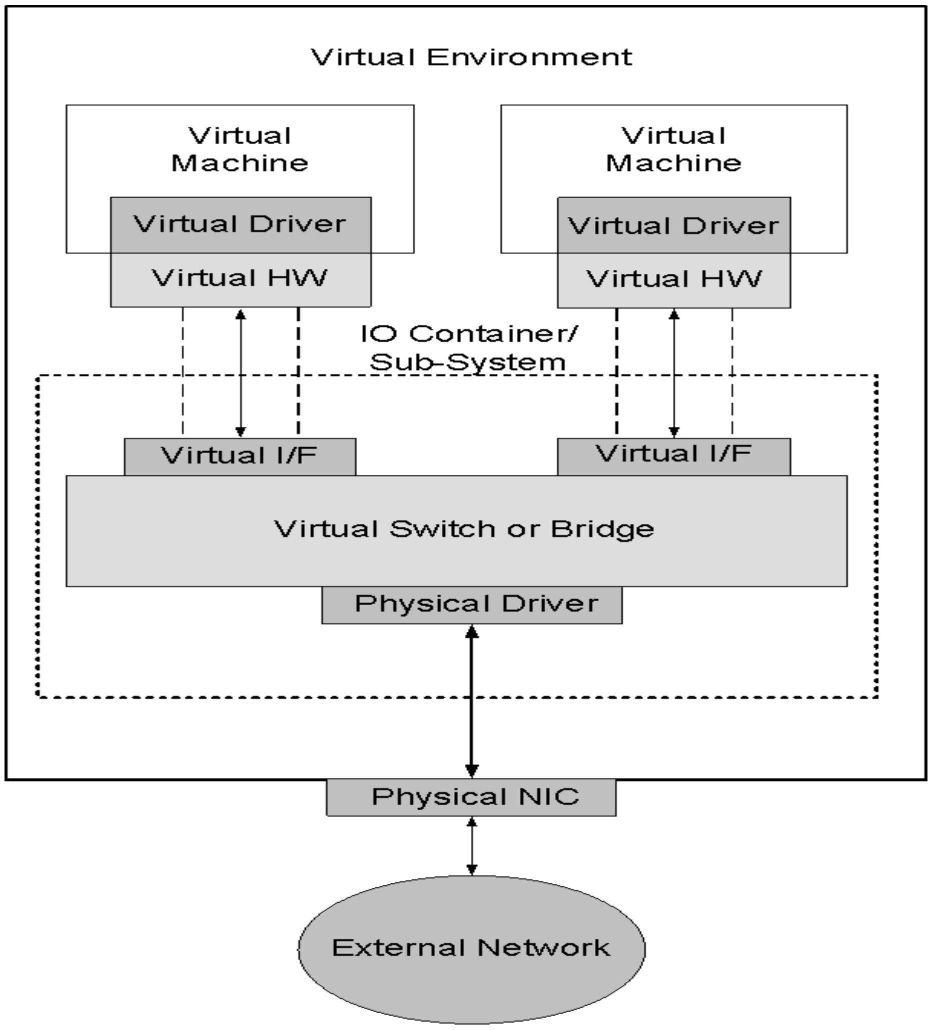

Figure 4. Emulated HW Sharing

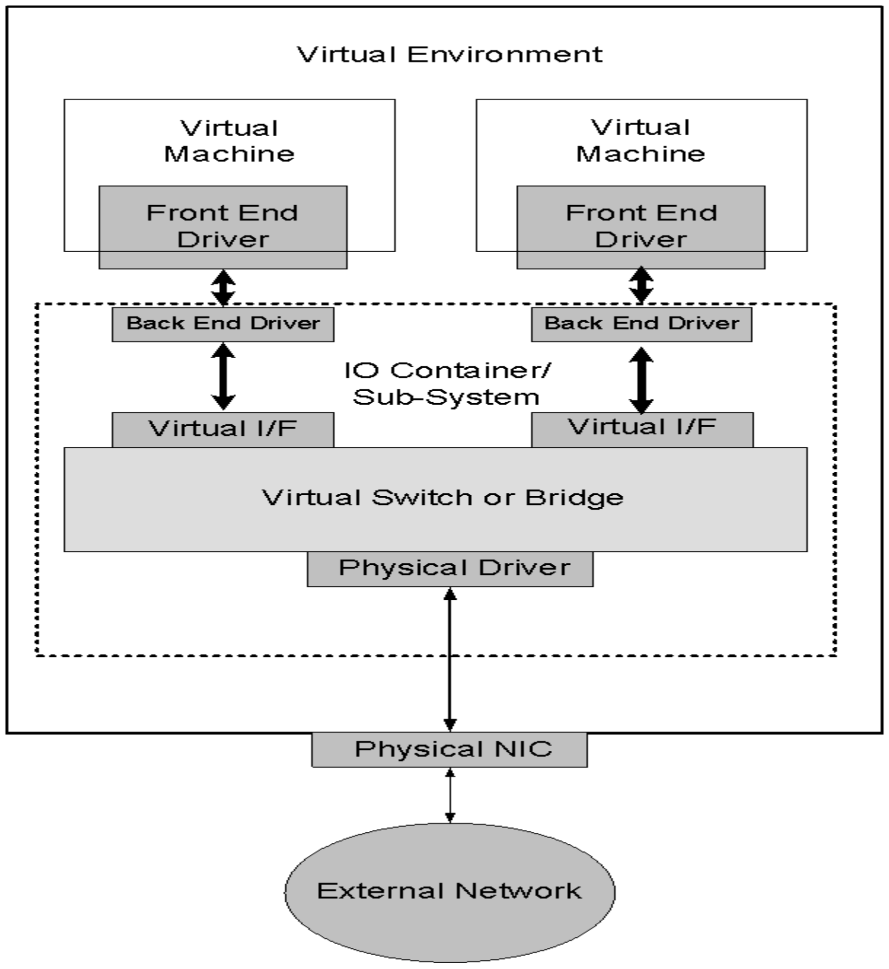

Figure 5. Front End / Back End Split Driver Sharing

2.3.1 Virtual Machine (VM)

Virtualization is the name for the creation of a number of execution environments on a single computer, each of which emulates a physical host computer. These execution environments are called **virtual machines (VMs)**. Each VM has its own operating system and resources (CPU, memory, storage, network).

2.3.2 I/O Container Models

An **I/O container/sub-system** (usually referenced as I/O container) is typically part of a privileged OS running in a persistent specialized VM which provides I/O services to other VMs (e.g., Xen's Domain0). A variant of the I/O container is an I/O sub-system whose software components are integrated into a VMM (e.g., VMware's ESX I/O architecture).

I/O containers provide abstractions of I/O subsystems so VMs can process I/O requests generated by guest software running in a VM. Two major I/O subsystems are the storage and networking subsystems. The focus of the VMDq architecture is to accelerate the network subsystem.

A typical I/O Container network subsystem is made up of several software components. The components include the physical network NIC driver, a virtual switch, and virtual network devices.

2.3.3 Virtual Driver

A **virtual driver** is typically a pseudo-Ethernet driver. Its upper interface is a standard OS-specific network driver interface. This allows an OS to send and receive packets through the virtual driver in the same manner as a non-virtualized Ethernet driver.

The virtual driver's lower interface is to virtual hardware or the split-driver model. This replaces the non-virtualized driver hardware interface.

2.3.4 Virtual Hardware

Virtual hardware is a virtual device that may be based on an existing physical device or a synthetic device. Virtual hardware emulates the behavior of a physical device (including PCI configuration). It appears as a standard network device to the Guest OS.

2.3.5 Split-Driver Model

The **split-driver model** takes a similar approach to the virtual driver model but instead of emulating a legacy device, the split-driver uses a **front-end driver** in the guest that works in concert with a **back-end driver** in an I/O partition. This model generally requires an I/O channel mechanism which allows the front-end driver to "communicate" to a back-end driver.

The back-end driver provides an "end point" that the front-end driver can "talk to". The driver is "hooked" to the virtual interface associated to the VM (in which the front-end driver is located).

2.3.6 Virtual Switch or Bridge

A key part of the I/O container network subsystem is the **virtual switch** or **virtual bridge** component. The terms virtual switch or virtual bridge (here referenced as virtual switch) are often used interchangeably to describe a layer-2 network device that is used to join LAN segments.

The virtual switch interconnects the virtual and physical LAN segments at the network interface layer (I.E. Layer 2) and forwards frames between them. The virtual switch acts as a MAC relay and is independent of any upper protocols.

The virtual switch is "transparent" to the Internet Protocol (IP) layer. For example, when a host sends an Ethernet frame to another host on a network connected by a virtual switch, the host sends the frame directly to the targeted host and the frame "crosses" the virtual switch without the sending or receiving hosts being aware of the virtual switch.

2.3.7 Virtual Interfaces

Virtual switches have **virtual interfaces**, which are enabled when VMs are created and bound to a specific virtual switch. Once bound, a VM is able to send or receive Ethernet frames using the virtual switch. The virtual interface "communicates" with a VM's virtual hardware by means of the VMM's **device channel** (a software mechanism).

Virtual Switch Physical Interfaces are added by a virtual switch control interface. The physical interface is bound to I/O container physical device drivers.

2.3.8 Physical Device Driver

The **physical device driver** (located in the I/O container) controls the physical network device. This is the I/O container operating system specific Ethernet driver that receives and sends Ethernet frames. The device driver interfaces with physical device to initialize the device; it controls resets, interrupts, and provides power management. An example of such a driver is the **NDIS miniport** under the Microsoft® Operating System and the **link driver** in Linux operating system.

The physical device driver upper interface is added to a virtual switch to allow VMs external network access.

3 VMDq Software Theory of Operations

A major goal in designing VMDq was to provide high performance I/O sharing in a manner that is sensitive to both hardware and software costs and complexities. To achieve this goal, the VMDq NIC re-purposes existing NIC filtering capabilities. In addition, VMDq was designed to minimize its impact to current I/O container network architectures to reduce the number of software changes required.

3.1 Current I/O Container Network Sharing

Current I/O containers provide a virtualized network infrastructure that is transparent to operating systems and network application software. This allows network applications to run unchanged in a virtualized environment. Classic virtualized network architecture is based around the emulation of a simple physical infrastructure consisting of physical host machines networked together using a physical network Layer 2 Switch. The network layout consists of host adapters connected to switch ports with the switch’s uplink port providing network access to external networks if needed.

An I/O container provides the network infrastructure with software components to connect virtualized systems together. This is accomplished using interfaces connecting physical and virtual devices to a virtual switch. This emulates a standard network such that networking applications can not tell they are running in a virtualized environment.

One of the main tasks of a virtual switch is to sort network traffic based on filters and to forward the traffic to the appropriate destination. The virtual switch is transparent to its communication partners. None of the host systems are aware that the virtual switch is present in the network path. To achieve transparency, the physical NICs used by virtual switches are configured to enable promiscuous mode. This allows the switch’s physical interfaces to receive incoming packets destined to any MAC address. The virtual switch inspects each packet’s destination address and, using layer 2 filters, decides whether a packet should be forward to another specific virtual switch interface or dropped.

Virtual switch interfaces can connect to physical devices or virtual devices. Virtual switch interfaces connected to virtual devices typically provide mechanisms that allow forwarded Ethernet frames to be directly/indirectly placed (using page flipping, page sharing, or CPU copy) into a VM’s virtual-device-receive buffers. This results in “sending” the forwarded packet from the virtual switch to a targeted VM. The Ethernet frame can be from another VM or from an external network (i.e, received by a physical network device).

A virtual switch allows VMs to transmit packets to external networks. A packet that is targeted to an external host will be forward to a virtual switch interface connected to a physical LAN device. Many LAN devices have only single transmit queue. This can be problematic when several VMs are sharing a physical LAN device. For example, a VM may transmit a large number of packets in a continuous burst. If another VM attempts to transmit a packet, the transmit must wait for previously queued transmit packets to “drain” from the LAN device’s single transmit queue. This is commonly referred to as ‘head of line’ blocking.

The I/O container model allows a single physical device to be shared by multiple VMs. I/O container implementations add additional overheads to both receive and transmit paths. The amount of overhead varies depending on the I/O container architecture.

3.2 I/O Container Network Sharing With VMDq

The performance of an I/O container can be improved by utilizing a VMDq device. A VMDq device improves the performance of the I/O Container by offloading some of the most commonly repeated I/O container tasks. VMDq architecture allows a VMDq device to offload the I/O container's virtual switch tasks including: packet sorting, moving data from the I/O container to the VM, routing packets to the most advantageous CPU core for receive processing, and support for transmit fairness.

3.3 VMDq Initialization

The default configuration of a VMDq adapter is to be in non-VMDq mode. When VMDq functionality is required, the driver must "enable" the adapter for VMDq mode. Depending on the VMDq implementation, this can include setting a VMDq control register and/or configuring a default queue. Refer to the applicable product's software developer's guide for details.

OS specific configuration mechanisms must be extended to add the ability to configure VMDq functionality (setting filters, identifying interrupts utilized by receive queues, mapping interrupts to processor cores, and to identifying "default" queues).

3.4 VMDq Receive Filter Setup

The process of categorizing packets into flows is accomplished by packet filtering. All packets matching a pre-defined filter are processed in the same configured manner by a VMDq device. For example, packets with the same Ethernet MAC destination address may be "classified" as a flow. Packet filters are required for the VMDq device to distinguish and isolate traffic into different flows for configured processing.

The following VMDq receive filter modes are supported:

- **Exact Unicast** — The destination address must exactly match a configured Unicast filter. Unicast filters will sort network packets and place them into the appropriate receive queues even if the adapter is configured to promiscuous mode.
- **Exact VLAN** — The Ethernet VLAN ID must exactly match a configured VLAN ID filter. VMDq devices do not replicate broadcast or multicast packets. If two or more VMs a receiving packets with the same VLAN ID, then software must provide a mechanism to replicate broadcast or multicast packets or prevent the use of VLAN ID filtering in such a network topology.
- **Promiscuous Mode** — Receive all unicast packets. A transparent virtual switch will set a VMDq adapter in promiscuous mode to allow the virtual switch to be transparent layer 2 switch. A VMDq adapter will filterer unicast/multicast packets using exact unicast/multicast address and place them into associated receive buffers. Packets not matching exact unicast addresses will be placed into the "default" queue.
- **Broadcast/Multicast** — Broadcast and multicast packets are placed in the 'default' queue when the VMDq adapter is in VMDq mode, regardless of VLAN filtering.

Normally, only good packets are received. These are defined as those packets with no CRC errors, symbol errors, sequence errors, length errors, alignment errors, or carrier extension errors. However, if the store–bad–packet bit is set, then bad packets that pass the filter function are stored in host memory. Packet errors are indicated by error bits in the receive descriptor. It is possible to receive all packets, regardless of whether they are bad, by setting the promiscuous enable and the store–bad–packet bit. Reception of bad packets is not a recommend practice.

3.5 VMDq Packet Pre-Sorting

By offloading an I/O container's virtual switch packet sorting task, the VMDq adapter provides the capability to filter received packets based on network Layer 2 criteria (MAC address). The sorted network traffic is then "sent" to the appropriate assigned receive queue.

VMDq software and hardware should be able to add and remove filters dynamically. Filters can be configured through the virtual switch or other I/O container specific mechanism. LAN drivers may add more filters to VMDq NIC hardware through OS specific implementations (if no OS API is available then a private method must be used).

VMDq architecture must have the ability to filter packets in promiscuous mode. Promiscuous mode is required by the virtual switch to allow transparent bridging. Any incoming packets "received" by the VMDq device in promiscuous mode will be checked against configured Layer 2 filters. If a packet matches a specific filter, then it will be placed into the receive queue assigned to that particular filter. If a packet does not match any filter, then it will be placed into a "default" queue to comply with the Virtual Switch "transparency" requirement.

3.5.1 VMDq Virtual Switch Packet Pre-Sort Receive Data Flow

The following is an example of a generic implementation of receiving a packet utilizing VMDq filtering capabilities:

- VMDq LAN device receives an unicast packet.
- VMDq device filters the packet against configured MAC address filters
  - If a filter matches packet destination MAC address, packet is placed into the associated receive queue.
  - If no filter matches packet destination MAC address, packet is placed into the 'Default' receive queue.
- The receive packet is DMA'd to the receive buffer as indicated by the receive queue descriptor.
- VMDq interrupts host with receive MSI-X (MSI or legacy) interrupt assigned to filter queue.
- VMDq physical device driver processes receive interrupt in OS specific manner.
  - If a matched filter receive queue reception:
    - Driver determines total number of received packets
    - Driver passes packets to OS specific receive network interface
    - Driver passes packets to OS specific receive interface (in group or one at a time with match indicator).
  - If a 'Default Receive Queue Reception
    - Driver passes packets to OS specific receive interface (in group or one at a time with non-match indicator).
    - Virtual Switch Physical Interface assigned to VMDq Physical Device receives packets either in a group or one at a time.
  - Receive Interface processes group of packets (as provided by receive packet interface or 'gathered' by Virtual Switch interface).

- If a group of packets with the same destination MAC address or single packet:
  - Virtual switch determines from one packet which virtual switch virtual interface to forward the group of packets.
  - Virtual switch forwards packet(s) as a group to appropriate virtual switch virtual interface.
  - Virtual switch virtual interface ‘moves’ packet(s) to bound VM’s virtual driver receive buffers.
  - I/O container’s receive buffer(s) are freed in OS specific manner.
  - Virtual switch virtual interface ‘notifies’ VM virtual driver of receive event.
  - VM virtual device driver processes received packet(s).

3.6 VMDq Transmit Fairness

A virtual switch provides access to external networks to each VM by supporting one or more interfaces that are connected to physical LAN devices. A well designed virtual switch should provide fairness between VM packet flows that contend for the same shared physical LAN transmit port. Each packet flow should get its fair share of the obtainable bandwidth and this share should not be affected by the presence and the behavior of other flows.

Unfortunately, many LAN devices have a single transmit queue that works in a FIFO manner. This makes it difficult to provide fairness between different VM flows. A common issue in a single queue device is ‘head of the line blocking’ in which a VM transmit must wait behind transmits generated by other VMs. A single misbehaving VM could consume an unfair share of LAN device bandwidth and impact the flows of all of machines in the host system. Assuming uniform distribution of transmit traffic between multiple VMs, ‘head of line blocking’ can reduce individual VM network throughput by as much as 59% of theoretical maximum.

A VMDq-capable LAN adapter provides hardware support to enable a **transmit fairness** solution. A VMDq-capable LAN device provides multiple transmits queues. Software utilizes these transmit queues to provide a solutions that will provide a measure of transmit fairness between VM packet flows.

VMDq LAN devices support a descriptor fetch arbiter and use a simple round robin scheduling scheme.

3.6.1 Transmit Fairness Data Flow

The following is an example of a generic implementation of transmitting packets using VMDq transmit fairness capabilities:

- A VM’s virtual Ethernet driver builds a transmit request.
- The VM virtual driver posts the transmit request to a VMM implementation specific ‘I/O channel’.
- The VM virtual driver notifies virtual switch virtual interface of the post transmit event (in a implementation specific manner).
- The virtual switch virtual interface removes indicated transmit request from ‘I/O channel’.
- The virtual interface builds an I/O container’s OS specific transmit request.
- The virtual interface passes OS specific transmit request for the virtual switch to forward to the appropriate virtual switch physical interface (if the transmit is targeted to an external network).
- The physical interface passes the transmit request to the VMDq LAN driver.

- The VMDq LAN driver places the transmit request packet on the appropriate transmit queue based on MAC address filtering criteria (note all packets of a given packet flow must be placed on the same transmit queue to prevent out of order packets).

3.7 VMDq Targeted Receives

A VMDq LAN device, because of its ability to place filtered packets in specific receive buffers, can further reduce operational receive overheads for an I/O container virtual switch using targeted receives. VMDq targeted receives require a VMM vendors to decide where in its architecture sorted received packets should be placed. For example, a VMM vendor could provide a design in which a VM shares its virtual driver receive buffers with the I/O container utilizing a VMDq LAN device.

The VMDq allocates receive buffers for non-default queues on a filter basis. For example, if a MAC address filter is set to 00-02-B3-90-DF-0E and this is the MAC address assigned to a VM, only that VM's shared buffers can be used with the 00-02-B3-90-DF-0E filter's receive queue (in targeted receive mode).

When a VMDq-enabled LAN device receives a packet with a destination address of 00-02-B3-90-DF-0E, the packet will be DMA'd to the shared buffers for a VM. In addition, the header is replicated. This allows the header to be passed to the virtual switch for processing. The virtual switch determines the virtual interface to which the partial packet should be sent and then forwards the header to that interface. The virtual interface indicates the packet reception to the virtual Ethernet driver in the VM. The VM Ethernet driver then recognizes that a received packet has been DMA'd into its receive buffer and can be processed.

Note that for a targeted receive to work with VT-d, shared buffers must be mapped to the I/O Container address space.

If a particular VM does not share its receive buffers with an I/O container, the I/O container receive buffers are used. Packets received matching the non-sharing VM filter would be placed into I/O container receive buffers. No header replication is needed with I/O container receive buffers.

Targeted receives may be used on some receive queues and not on others.

3.7.1 VMDq Targeted Receive Data Flow

The following is an example of a generic implementation of receiving a packet utilizing VMDq targeted receive capabilities:

- The VM virtual Ethernet driver allocates receive buffers.
- The virtual Ethernet driver posts and indicates that the receive buffers are to be shared with the I/O container.
- The I/O container virtual interface 'removes' shared receive buffers indicated by VM virtual driver.
- The I/O Container virtual interface requests the VMM to validate shared receive buffers.
- The VMM checks to see if the VM can share the indicated pages.
    - If the pages are valid, pages are mapped into the I/O container if in a VT-d enabled environment.
    - If the pages are not valid, the VMM fails the call and returns an error.

- The I/O container virtual interface places shared receive buffers into filter specific allocation pools.
- The I/O container VMDq LAN driver allocates shared receive buffers from appropriate filter allocation pool of VM shared receive buffers (if the VM has not shared any receive buffers the I/O container receive buffers are used).
- VMDq LAN device receives a unicast Ethernet packet.
- VMDq device matches the packet's destination address against filters.
- The received packet is placed into the corresponding filter receive queue or if no matching filter default queue.
- VMDq device DMAs the packet to the associated receive queue receive buffer.
  - If the receive packet matches a queue with VM shared buffers, its header is replicated and DMA'd to the I/O container.
- The VMDq LAN device interrupts the host with the MSI-X interrupt assigned to the receive filter.
- VMDq physical device driver processes the receive interrupt event in an OS-specific manner.
- VMDq LAN driver determines number of received packets in the filter queue.
- VMDq LAN driver passes replicated headers of filter matching packets to OS-specific receive interface (in a group); it indicates to the OS in group or one at a time with match indicator.
- Virtual switch physical interface bound to the VMDq physical device receives packets.
  - If in a group of sorted matched packets or single packet:
    - Virtual switch interface determines from passes packet OS-specific receive information if receive is targeted receive.
      - If targeted receive:
        - The interface indicates receive event to virtual switch virtual interface connected to VM that originally shared the receive buffer.
        - The virtual switch virtual interface relinquishes control of the shared receive buffers.
        - The VM virtual Ethernet driver process receive event using previously shared receive buffers.
      - If not:
        - Process packets in virtual switch pre-sorted manner.

4 Ethernet Controller Support

The following Intel® Ethernet Controllers support Intel VMDq features.

Intel® Ethernet Controller	Number of Queue Pairs (Per Port)
Intel® 82575 Gigabit Ethernet Controller	4
Intel® 82598 10 Gigabit Ethernet Controller	16

5 References

Intel® Virtualization Technology for Directed I/O Architecture Specification, February 2006.

Intel® Virtualization Technology Specification for IA-32 Intel® Architecture, April 2005. Mark

Brunstad. "Networking Virtual Machines Under ESX Server: Part 1," Session PAC195-A. *VMWorld,* 2005.

Howie XU. "Networking Virtual Machines Under ESX Server: Part 2, Networking Internals Session PAC195-B". *VMWorld,* 2005.

Brian Henry. "Network Virtual Service Provider Deep Dive". *Microsoft Windows Virtualization Design Preview,* December 2005.

"Virtual Machine Device Queues: An Integral Part of Intel® Virtualization Technology for Connectivity that Delivers Enhanced Network Performance," White Paper, Intel Corporation.