Receive-Side Scaling Enhancements in Windows Server 2008

November 5, 2008

Abstract

This paper provides information about receive-side scaling (RSS), a technology that enables packet receive-processing to scale with the number of available computer processors. This paper provides an overview of RSS for NDIS driver developers and discusses the implications of RSS for driver and hardware development. The paper also describes the new enhancements introduced in Windows Server® 2008 that implement NDIS v6.1 for RSS coexistence with PCI v3.0 message signaled interrupts (MSI-X).

Original equipment manufacturers (OEMs) and IT administrators are also encouraged to read this paper to gain better understanding of RSS and how it is implemented in Windows Server 2008.

The information in this paper applies to the following operating systems:

Windows Vista SP1
Windows Server 2008

References and resources discussed here are listed at the end of this paper.

For the latest information, see:
 <http://www.microsoft.com/whdc/device/network/NDIS_RSS.mspx>

**Disclaimer:** This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Change |  |  |  |
| November 5, 2008 | First publication |

Contents

[Introduction 3](#_Toc213142752)

[Packet Receive-Processing Limitations without RSS 3](#_Toc213142753)

[Packet Receive-Processing with RSS 4](#_Toc213142754)

[RSS Algorithm 4](#_Toc213142755)

[RSS Versus non-RSS Receive Processing 4](#_Toc213142756)

[RSS Setup 6](#_Toc213142757)

[RSS Capabilities Advertisement 6](#_Toc213142758)

[Configuring the RSS Parameters 7](#_Toc213142759)

[Selection of CPUs Eligible for RSS 8](#_Toc213142760)

[Selection of the Default RSS Hash Function 9](#_Toc213142761)

[Toeplitz Hash Function Specification 9](#_Toc213142762)

[Mapping Packets to Processors 11](#_Toc213142763)

[Packet Receive Processing with RSS 12](#_Toc213142764)

[RSS Implementation 12](#_Toc213142765)

[RSS and MSI-X 13](#_Toc213142766)

[Example RSS NIC with MSI-X Capability 13](#_Toc213142767)

[RSS Load-Balancing Implementation in Windows Server 2008 16](#_Toc213142768)

[RSS Configuration Parameters 16](#_Toc213142769)

[RSS Limitations 17](#_Toc213142770)

[Resources 18](#_Toc213142771)

# Introduction

Today’s systems have an increasing number of CPUs. The ability of the networking protocol stack of the Windows® operating system to scale well on a multi-CPU system is restricted. This restriction is caused by the architecture of the *Network Driver Interface Specification* (NDIS) in Windows Server® 2003 and earlier versions, which limits receive protocol processing to a single CPU at any one time. Receive-side scaling (RSS) resolves this issue by allowing the network load from a network adapter to be distributed across multiple CPUs.

This paper is for the technical community that wants to gain deeper insight into how RSS operates in Windows Server 2008. It provides specific insights into implementation issues for independent hardware vendors (IHVs), and for original equipment manufacturers (OEMs) and system administrators who want to understand how the technology works.

Windows Server 2003 SP1 and earlier versions allows only a single deferred procedure call (DPC) for each network adapter to execute at any one time. Windows Server 2003 SP2 and newer versions of Windows Server that use RSS enable multiple DPCs on different CPUs for each instance of a network adapter miniport driver, while they preserve in-order delivery of messages on a per-stream basis. RSS also supports dynamic load balancing, a secure hashing mechanism, parallel interrupts, and parallel DPCs.

The information in this paper applies to Windows Server 2008 and Windows Vista® SP1 unless otherwise noted.

## Packet Receive-Processing Limitations without RSS

Windows Server 2003 SP1 and earlier versions do not allow multiple processors to concurrently process receive indications from a single-network adapter. NDIS version 5*.x* was included with Windows Server 2003 SP1 and earlier versions. In this version, a packet that is received from the network on a specific network adapter manifests itself as an interrupt to the host processor from the network adapter and eventually causes a DPC to be queued on one of the system processors. The DPC runs to completion, typically on the processor that hosted the interrupt, and additional interrupts from the network adapter are disabled until the DPC completes its cycle.

Many scenarios, such as large file transmissions, require the host protocol stack to perform significant work in the context of receive DPC processing (for example, sending out new data or performing memory copy). In these scenarios, the lack of parallelism in NDIS v5*.x* packet receive processing results in an overall lack of scaling.

In addition, some contemporary CPUs and chipsets route all interrupts from a single network adapter to one specific processor, which results in a similar lack of parallelism. Therefore, scaling issues only increase because one CPU handles all device interrupts.

## Packet Receive-Processing with RSS

The single-CPU processing issues are resolved by implementing RSS. This technology enables receive processing to be balanced across multiple processors in the system while in-order delivery of the data is maintained. RSS enables parallel DPCs. In addition, in Windows Server 2008 and later versions if the computer and network adapter support it, RSS enables parallel interrupts.

RSS provides the following benefits:

* Parallel receive processing.

Receive packets from a single network adapter can be indicated by generating interrupts and DPCs concurrently on multiple CPUs.

* Preserving in-order packet delivery.

Received packets for a specific stream from a single network adapter are delivered in order to the TCP/IP protocol driver.

* Dynamic load balancing.

As system load on the host varies, RSS rebalances the network processing load among the processors.

* Cache locality.

Because packets from a single connection are mapped to a specific processor, state for a particular connection stays resident in the cache of the processor. This eliminates cache thrashing and also improves performance.

* Send-side scaling.

The TCP/IP protocol passes the RSS hash value to the NIC in each packet on the egress path, which allows the send completions to be indicated on the same CPU. This enables better scaling on the send side.

* Toeplitz hashing.

The default generated RSS signature is statistically secure. This makes it more difficult for malicious remote hosts to force the system into an unbalanced state.

# RSS Algorithm

This section defines the RSS algorithm and contrasts it with the non-RSS packet processing algorithm. Generally, RSS enables packets from a single network adapter to be processed in parallel on multiple CPUs while it preserves in-order delivery to TCP connections.

## RSS Versus non-RSS Receive Processing

The NDIS 5.1 architecture for processing incoming packets is implanted in Windows Server 2003 and earlier versions. A network adapter vendor typically implements this architecture by taking advantage of a receive descriptor queue between the network adapter and the miniport adapter to pass per-packet information. The packets are processed in the following sequence:

1. As packets arrive off the wire at the network adapter, the packet contents are transferred into host memory by using direct memory access (DMA), and a receive descriptor is transferred into the receive descriptor queue (again through DMA). An interrupt is eventually posted to the host to indicate that new data is present. Exactly when the interrupt fires depends on the interrupt moderation scheme.

2. Depending on the system’s chipset and CPUs, either the interrupt is distributed to one of the host processors or it is always routed to the same processor.

3. If additional packets arrive at the network adapter, then data and descriptors are transferred to host memory by using DMA. An interrupt is not fired.

4. The interrupt service routine (ISR) runs on the host processor to which the interrupt was routed, which disables further interrupts from the network adapter. The ISR then schedules the miniport adapter’s DPC to run on a specific processor—usually the same processor that was used to run the ISR, unless the DPC is explicitly set to run on another processor.

5. When the DPC runs, it processes the receive descriptor queue. Either the DPC creates a list of packets to hand to the NDIS interface, or it signals each packet to the NDIS interface, one at a time. In either case, no other processor can perform network adapter interrupt processing because interrupts from the network adapter are disabled.

6. The protocol stack processes each indicated packet. For TCP, this involves updating internal state, potentially sending new data if the TCP window allows it to do so, and potentially indicating or completing data to the application.

7. After all receive descriptors are consumed or some maximum amount of processing is done, the DPC reenables interrupts on the network adapter and returns. This action allows another interrupt to be triggered on another (potentially different) host processor.

RSS enables parallelism by changing steps 5 and 7 to allow the following algorithm to be implemented:

Fire multiple ISRs to specific processors that cause multiple DPCs to be scheduled in parallel. As shown in step 4, a specific interrupt remains disabled and is reenabled only after a single DPC (or group of DPCs for a given ISR) has executed in step 7.

The described sequence of events enables parallel processing of received packets. However, if in-order delivery is not preserved, performance will probably be degraded. For example, if packets for a group of connections are processed on different CPUs and one CPU is lightly loaded while the other is heavily loaded, older packets could actually be processed first. Because the generation and processing of TCP acknowledgment are highly optimized for in-order processing, performance is degraded unless RSS supports in-order delivery of TCP segments.

RSS enables in-order packet delivery by ensuring that one processor processes packets for a single TCP connection. This RSS feature requires that the network adapter examine each packet header and then use a hashing function to compute a signature for the packet. To ensure that the load is balanced evenly across the CPUs, the hash result is used as an index into an indirection table. Because the indirection table contains the specific CPU that is to fire the interrupt and run the associated DPC and the host protocol stack can change the contents of the indirection table at any time, the host protocol stack can dynamically balance the processing load on each CPU.

Figure 1 shows the RSS processing sequence. As shown on the right side of Figure 1, incoming network packets arrive for processing. The hash function is applied to the header to produce a 32-bit hash result. The hash type controls which incoming packet fields are used to generate the hash result. The hash mask is applied to the hash result to get the number of bits that are used to index in the indirection table. The indirection table contains CPUs that are used in RSS. The lookup in the indirection table identifies the CPU that the network adapter uses to indicate the received packets to the operating system.

Because the host protocol stack must change the processing load on each processor by varying the contents of the table, the size of the table in Figure 4 must be significantly larger than the number of CPUs in the system.

Figure 1. RSS receive-processing sequence in the network adapter hardware

## RSS Setup

The process of initializing RSS consists of the following two steps:

* Advertisement of RSS capabilities by the miniport driver that is associated with the network adapter to NDIS.
* Configuration of the RSS parameters that are used by the network stack (TCP/IP and NDIS) and the network adapter.

The next two sections describe each step in more detail.

### RSS Capabilities Advertisement

In Windows Server 2008, which implements NDIS v6.1, the advertisement of the RSS capabilities by the network adapter happens when the NDIS miniport driver is initialized with NDIS. The RSS capabilities are passed by the driver to NDIS in the [NDIS\_MINIPORT\_ADAPTER\_GENERAL\_ATTRIBUTES](http://msdn.microsoft.com/en-us/library/bb743053.aspx) structure as described in the “[Initializing a Miniport Driver](http://msdn.microsoft.com/en-us/library/ms795246.aspx)” section of the Windows Driver Kit (WDK). During this step, the miniport driver reports the hashing functions and hashing types that it supports to NDIS. For more detailed information see the “[RSS Configuration](http://msdn2.microsoft.com/en-us/library/ms795612.aspx)” section in the WDK.

Windows Server 2008 introduced support for PCI v2.2 MSI and PCI v3.0 MSI-X. Network adapters that support RSS are strongly encouraged to also support MSI-X to be able to distribute the device interrupts across the set of RSS CPUs.

**The miniport driver should allocate MSI-X resources for the device during driver initialization time as specified in the “**[MSI-X Pre-Registration](http://msdn.microsoft.com/en-us/library/bb961705.aspx)” and “[Registering and Deregistering an MSI Interrupt](http://msdn.microsoft.com/en-us/library/bb961699.aspx)” section of the Windows Server 2008 WDK. During the MSI-X resource allocation, the driver should allocate the same number of MSI-X messages as there are RSS CPUs in the system.

**After this initialization phase is completed, the driver can, at run time, assign an MSI‑X table entry to *any one* CPU. The miniport driver can** mask, unmask, or map MSI-X table entries to device-assigned MSI-X messages at run time. For more details, see section “[MSI-X Resource Filtering](http://msdn.microsoft.com/en-us/library/bb961708.aspx)” in the Windows Server 2008 WDK.

### Configuring the RSS Parameters

After the miniport driver is initialized and NDIS is aware of the miniport’s RSS capabilities, the TCP/IP protocol driver configures the RSS parameters through NDIS. The following variables are the main RSS parameters that the TCP/IP protocol driver configures. For example, *4-tuple* means four parameters are used, and *2-tuple* means that two parameters are used:

* Hash function.

The default hash function is the Toeplitz hash. No other hash functions are currently defined.

* Hash type.

The hash type is the fields that are used to hash across the incoming packet. Depending on what the miniport adapter advertises that it can support, the host protocol stack can enable any combination of the following set of flags:

4-tuple of source TCP Port, source IP version 4 (IPv4) address, destination TCP Port, and destination IPv4 address.

4-tuple of source TCP Port, source IP version 6 (IPv6) address, destination TCP Port, and destination IPv6 address.

2-tuple of source IPv4 address and destination IPv4 address.

2-tuple of source IPv6 address and destination IPv6 address.

2-tuple of source IPv6 address and destination IPv6 address, including support for parsing IPv6 extension headers.

For additional information about combining hash field flags, see the “[RSS Hashing Types](http://msdn2.microsoft.com/en-us/library/ms795613.aspx)” section of the WDK.

* Hash bits (or mask).

The hash bits are the number of hash-result bits that are used to index into the indirection table. All network adapters must support 7 bits. The host protocol stack sets the actual number of bits to be used during initialization. The number will be between 1 and 7, inclusive. This range effectively defines the size of the indirection table.

* Indirection table.

An indirection table is the data structure that contains an array of CPU numbers to be used for RSS. The host protocol stack periodically rebalances the network load by changing the values in the indirection table.

* Secret hash key.

The size of the key depends on the hash function. For the Toeplitz hash, the size is 40 bytes for IPv6 and 16 bytes for IPv4.

After RSS is initialized, data transfer can begin. Over time, the host protocol stack (TCP/IP) calls the [OID\_GEN\_RECEIVE\_SCALE\_PARAMETERS](http://msdn2.microsoft.com/en-us/library/bb892680.aspx) configuration object identifier (OID) to modify the indirection table to rebalance the processing load. Usually, all parameters in the OID are the same except for the values in the indirection table. However, after RSS is initialized, the host protocol stack may change other RSS initialization parameters. This occurrence is extremely rare, so it is acceptable to require a hardware reset to change the hash algorithm, the secret hash key, the hash type, or the number of hash bits used.

## Selection of CPUs Eligible for RSS

Careful selection of processors that should be used for RSS is an important aspect of the RSS load balancing algorithm. TCP/IP and NDIS strive to select CPUs for RSS purposes that do not reside on the same core, which avoids the use of hyperthreading CPUs for RSS purposes.

The CPU numbers that are used for RSS vary for different platforms, operating system versions, instruction set architectures (x86 vs. x64 vs. IA‑64) and, on Intel CPUs, whether Hyperthreading is enabled. For example, Figure 2 shows the CPU numbering for a dual-socket system that is running 32-bit Windows Server 2008. Figure 3 shows the same system running 64-bit Windows Server 2008. Notice that the CPU numbering is different, depending on whether an x86 or x64 bit version of Windows is running on the system.



Figure 2. CPU numbering in dual-socket system running x86 Windows Server 2008



Figure 3. CPU numbering in dual-socket system running x64 Windows Server 2008

The CPUs selected for RSS are always derived from the list of CPUs that [**NdisGetProcessorInformation()**](http://msdn.microsoft.com/en-us/library/bb500939.aspx)returns. NDIS miniport drivers must use this device driver interface (DDI) to obtain information about the CPU topology of the system for the purposes of RSS.

## Selection of the Default RSS Hash Function

The default RSS hash function was chosen after considerable research into hash algorithms, by using the following criteria to evaluate candidate hash functions:

* How evenly distributed the hash was for different hash inputs and different hash types, including TCP/IPv4, TCP/IPv6, IPv4, and IPv6 headers. For each hash algorithm, simulations were performed to analyze the resulting hash randomness. Different inputs to the hash functions were used, including empirical data from different classes of servers and a function that produced random numbers.
* How evenly distributed the hash was when only a few buckets were present (for example, two or four).
* How randomly distributed the hash was when many buckets were present (for example, 64 buckets).
* How computationally intensive the hash calculation was for software.
* How easy it was to implement the hash in hardware. This was determined through discussions with network adapter vendors.
* How difficult it would be for a malicious remote host to send packets that would all hash to the same bucket, thus eliminating the advantages of RSS.

The Toeplitz hash was selected as the base algorithm that all RSS implementers must support, with a programmable number of buckets that can vary from 2 to 128.

## Toeplitz Hash Function Specification

The following four pseudocode (p-code) examples show how to calculate the NdisHashFunctionToeplitz hash value for different packet fields.

To simplify the examples, a generalized algorithm that processes an input byte-stream is used. Specific formats for the byte-streams are defined in each example. The overlying driver provides a secret key (K) to the miniport adapter that is used in the hash calculation. The key is 40 bytes (320 bits) long for the Toeplitz hash on IPv6 and 16 bytes (128 bits) for IPv4 packets. For sample data that enables a hardware vendor to verify its implementation, see the “[Verifying the RSS Hash Calculation](http://msdn2.microsoft.com/en-us/library/ms795606.aspx)” section in the WDK.

Given an input array that contains *n* bytes, the byte stream is defined as follows:

input[0] input[1] input[2] … input[n-1]

The leftmost byte is input[0], and the most significant bit of input[0] is the leftmost bit. The rightmost byte is input[n-1], and the least significant bit of input[n-1] is the rightmost bit.

Given the preceding definitions, the p-code for processing a general input byte stream is defined as follows:

ComputeHash(input[], n)

result = 0
For each bit b in input[] from left to right {

if (b == 1) result ^= (left-most 32 bits of K)
shift K left 1 bit position

}
return result

The p-code contains entries of the form @n-m. These entries identify the byte range of each element in the TCP packet, assuming an Ethernet DIX packet format is used. The result represents the final value of the hash, which is 32 bits.

#### Example: Hash Calculation for IPv4 with the TCP Header

Concatenate the SourceAddress, DestinationAddress, SourcePort, and DestinationPort packet fields into a byte array, preserving the order in which they occurred in the packet:

Input[12] = @12-15, @16-19, @20-21, @22-23

Result = ComputeHash(Input, 12)

#### Example: Hash Calculation for IPv4 Only

Concatenate the SourceAddress and DestinationAddress packet fields into a byte array:

Input[8] = @12-15, @16-19

Result = ComputeHash(Input, 8)

#### Example: Hash Calculation for IPv6 with the TCP Header

Concatenate the SourceAddress, DestinationAddress, SourcePort, and DestinationPort packet fields into a byte array, preserving the order in which they occurred in the packet:

Input[36] = @8-23, @24-39, @40-41, @42-43

Result = ComputeHash(Input, 36)

#### Example: Hash Calculation for IPv6 Only

Concatenate the SourceAddress and DestinationAddress packet fields into a byte array:

Input[32] = @8-23, @24-39

Result = ComputeHash(Input, 32)

## Mapping Packets to Processors

To generate a 32-bit signature, RSS requires the hash function to be calculated over a specific set of fields in the packet header. The set of fields over which the hash is computed is set by the configuration OID. If the hash type flags enable only one type of hash, then any packet that is received that does not match that hash type is not hashed. For definitions of hash type flags, see “RSS Initialization” earlier in this paper and “[RSS Hashing Types](http://msdn2.microsoft.com/en-us/library/ms795613.aspx)” in the WDK.

For example, if the TCP/IPv4 4-tuple hash type is set and a non-TCP packet arrives—for example, an IP fragment, a User Datagram Protocol (UDP) packet, or an IPsec-encrypted packet—a hash is not performed on the packet. If multiple flags are set—for example, if the TCP/IPv4 and IPv4 hash types are enabled—then if the packet is not a TCP/IPv4 packet but is an IPv4 packet—for example, an IPsec-encrypted packet, the hash is performed on just the IPv4 2-tuple. Further, for this setting of the hash type flags, if the incoming packet is not an IPv4 packet, then no hash is performed. Because many different hash types can be applied on a per-packet basis (including no hash), the hash type is indicated to the host protocol stack on a per‑packet basis. If no hash was performed, then none of the hash type flags is set.

As mentioned earlier, the hash signature for each packet is masked and then used to index into the indirection table to determine the processor on which the packet should be processed (that is, the processor that runs the ISR and DPC). Figure 4 shows a specific example of the contents of the indirection table and how it can be changed, assuming that a four-processor configuration is used and that the configuration OID sets the number of least significant bits for the hash result mask to be 6 bits. This configuration implies that the indirection table contains 64 entries.

In Figure 4, Table A shows the indirection table values immediately after initialization. After some time has elapsed and as regular traffic load varies, the processor load grows unbalanced. The host protocol stack then detects the unbalanced condition and tries to rebalance the load by calculating a new indirection table and passing it to the miniport driver by using the configuration OID. Table B in Figure 4 shows the new indirection table, which moved some load off CPU 2 and onto CPUs 1 and 3.

When the indirection table is changed, it is possible for a short time (while the current receive descriptor queues are being processed) for packets to be processed on the wrong CPU. This is an acceptable transient condition that should last as long as the NIC hardware and driver take to drain a hardware queue full of received packets. During this transient condition the performance, as measured by CPU usage and throughput, might degrade.



Figure 4. Example: RSS indirection table shown in Table A; Table B shows the indirection table after some receive processing is moved away from CPU 2

The RSS implementation in NDIS 5.2 (Windows Server 2003 SP2) had limitations in handling some degenerative cases that are common when you are running benchmarks. For example, the load balancing of a single, very active TCP connection is not optimal because the TCP connection is moved across CPUs too aggressively, which causes some decrease in performance.

The load balancing implementation of RSS in Windows Server 2008 addresses the previously mentioned limitation.

## Packet Receive Processing with RSS

NDIS data reception with RSS is started by the [**MiniportMessageInterrupt**](http://msdn.microsoft.com/en-us/library/bb961876.aspx) ISR being scheduled to run on a specific CPU. Because of the ISR routine running, NDIS queues a DPC—[**MiniportMessageInterruptDPC**](http://msdn.microsoft.com/en-us/library/bb961879.aspx) —to run on the same CPU where the ISR ran. When the DPC is ready to indicate packets to the host protocol stack, through a call to [**NdisMIndicateReceiveNetBufferLists**](http://msdn.microsoft.com/en-us/library/ms800382.aspx), it initializes the following fields in the NetBufferList data structure:

* RSS hash function; see “[NET\_BUFFER\_LIST\_SET\_HASH\_FUNCTION](http://msdn2.microsoft.com/en-us/library/bb892675.aspx).”
* RSS hash type (if no hash was calculated, then none of the hash type flags is set); see “[NET\_BUFFER\_LIST\_SET\_HASH\_TYPE](http://msdn2.microsoft.com/en-us/library/bb892676.aspx).”
* RSS hash value (including all 32 bits); see “[NET\_BUFFER\_LIST\_SET\_HASH\_VALUE](http://msdn2.microsoft.com/en-us/library/bb892673.aspx).”

For more detailed description see the “[Indicating RSS Receive Data](http://msdn2.microsoft.com/en-us/library/ms795610.aspx)” section in the WDK.

# RSS Implementation

When a network adapter supports the RSS hash calculation in hardware, many implementation options are possible. This section describes the optimal RSS implementation from scalability and performance point of view.

Network adapter vendors should incorporate hardware-specific tradeoffs, cost tradeoffs, and limitations that are imposed by the host system implementation when they are designing RSS-capable network adapters. The information in this section is intended to be a high-level example, and implementation issues may force different implementation options.

## RSS and MSI-X

The optimal RSS implementation would be for the network adapter to support PCI v3.0 MSI-X[[1]](#footnote-2) and to have as many hardware receive queues as there are CPUs in the system. With such a network adapter the sequence of events on the receive (ingress) path is as follows:

1. At initialization time each hardware queue is associated with a CPU on the system.

2. At run time, as packets arrive off the wire, the hash fields are selected according to the hash type, the RSS hash is calculated, and the hash mask is applied to the result. The result is used to find the correct hardware queue in which the packet should be placed.

3. The packet contents are transferred into host memory by using DMA writes and a receive descriptor is transferred into the receive descriptor queue by means of a DMA write.

4. The network adapter uses MSI-X to directly interrupt the processor that is associated with the specific hardware queues and indicate the presence of new data. Exactly when the interrupt fires depends on the network adapter’s interrupt moderation scheme, which should account for RSS.

5. If additional packets arrive at the network adapter, data and descriptors are transferred by means of DMA to host memory.

6. The ISR runs on the host processor to which the interrupt was routed. The ISR processes the interrupt request, disables the firing of additional interrupts on the specific CPU, and schedules a DPC to execute.

7. When the DPC runs, it process the receive descriptors on the receive descriptor queue and indicates the set of packets up to the NDIS interface, which preserves the packet order in the receive descriptor queue.

One complication that the hardware and driver developer must handle is the additional level of indirection—TCP connection <-> Hardware Queue <-> CPU, where the hardware queue to CPU mapping is not one to one.

In Windows Server 2008, the RSS implementation does not consider the number of hardware queues that the NIC has when assigning CPUs for RSS purposes. This implies that the network adapter can be asked to deliver packets to more CPUs than it has hardware queues for. The network adapter and driver must be able to handle this situation.

The following sections describe how a network adapter and driver developer can handle this situation.

## Example RSS NIC with MSI-X Capability

Figure 6 shows a network adapter that has 4 hardware queues and 16 MSI-X table entries and is plugged in a system that has 4 CPUs allocated for RSS. The hardware can have more than four available receive queues, but because the system is currently using only four processors for RSS, only four RSS queues are being used. This is an example of a trivial RSS configuration because one hardware queue exists for each RSS CPU in the system.

Figure 5 shows the indirection table passed by the TCP/IP protocol driver to the network adapter that is shown in Figure 6.

Figure 5. Indirection table populated with four CPUs



Figure 6. Network adapter with four hardware queues and five MSI-X table entries

In Figure 6, note that the ”Hash-To-Queue Table” is equal to the RSS indirection table that was passed to the miniport driver by the TCP/IP protocol driver that is shown in Figure 5.

Figure 8 shows a hardware and software configuration of an MSI-X-capable RSS NIC with a smaller number of hardware queues (two) than RSS processors (four). Furthermore, the system administrator has reserved CPU 0 for application use and therefore the RSS indirection table passed to the driver starts with CPU 1.

Figure 7 shows the indirection table passed by the TCP/IP protocol driver to the network adapter that is shown in Figure 8.

Figure 7. Indirection table with four CPUs starting with CPU number 1



Figure 8. Network adapter with four hardware queues and five MSI-X table entries

The network adapter in Figure 8 can choose to handle delivering received packets to any one of the four CPUs even though it has only two hardware queues. This implies that the driver must allocate MSI-X resources for all four CPUs. **Then, at run time, the miniport driver must change the CPU affinity of the MSI-X table entries so that it delivers packets from the two hardware queues to all four CPUs.**

**We do not recommend this implementation approach, which can adversely affect the performance of the system. W**hen the NIC has less hardware queues and MSI-X table entries than CPUs that are used for RSS, the miniport driver—potentially frequently—updates the CPU affinity of a specific MSI-X table entry. Each MSI-X table entry update triggers a series of PCI register writes that are slow operations when they are compared to the CPU speed. The final effect of this behavior is degradation in performance (as measured in throughput) during these MSI-X table entry updates.

A more appropriate solution to the problem of dealing with fewer hardware queues than RSS CPUs is for the RSS implementation in the driver to limit the number of RSS CPUs that are used for the network adapter. This can be achieved by the NIC driver selecting a subset of RSS CPUs that were passed to it in the RSS indirection table. Figure 8 demonstrates how this can be achieved. The “Hash-To-Queue Table” that the hardware uses is a subset of the RSS indirection table that is shown in Figure 7.

Device driver developers must make sure that the ”Hash-To-Queue Table” is always synchronized with the RSS indirection table to the extent that the former always contains a subset of the entries in the indirection table.

## RSS Load-Balancing Implementation in Windows Server 2008

The RSS load-balancing implementation strives to ensure that no single CPU becomes a bottleneck for processing more ingress networking traffic. The RSS load-balancing implementation does not strive to distribute the ingress network processing load *evenly* across the set of RSS CPUs. Therefore, it is acceptable for a system with four CPUs to have a CPU load of 70, 10, 10, and 10 percent across the four RSS CPUs.

The TCP/IP protocol driver (TCPIP.SYS) implements the network load balancing. TCP/IP implements the load balancing by sending indirection table updates to the NIC driver through NDIS. To achieve the previously mentioned goal, TCP/IP uses the following inputs when it makes load-balancing distribution decisions:

* Send and receive I/O load per group of TCP connections.
* Send and receive I/O load per CPU per group of TCP connections.
* Individual CPU usage.

The TCP/IP protocol driver periodically examines the input parameters and determines whether to update the indirection table. The TCP/IP protocol driver has a 2‑second timer that triggers the examination of the current load distribution. In addition, TCP events such as TCP connection establishment trigger the examination of the current load and update the indirection table if needed.

## RSS Configuration Parameters

In Windows Server 2008, the system default is for RSS to be enabled and to use a maximum of four CPUs. OEMs and administrators can adjust the maximum number of RSS processors by setting the **MaxNumRssCpus** registry value in HLKM\System\CurrentControlSet\Services\Ndis\Parameters. The value type is DWORD and if not present, the default value of four processors is assumed. Note that this value *must not* be set to value greater than 32.

In Windows Server 2008, an administrator can set the base CPU number that should be used for RSS. You can do this by setting the **RssBaseCpu** registry value in HLKM\System\CurrentControlSet\Services\NDIS\Parameters. The value type is DWORD and, if not present, a default value of 0 is assumed.

In Windows Server 2008, an administrator can determine the current state of RSS (enabled/disabled) in the operating system by executing the following command from an elevated command prompt:

C:\>netsh int tcp show global

Querying active state...

TCP Global Parameters

----------------------------------------------

Receive-Side Scaling State : enabled

Chimney Offload State : enabled

Receive Window Auto-Tuning Level : highlyrestricted

Add-On Congestion Control Provider : none

ECN Capability : disabled

RFC 1323 Timestamps : disabled

To enable or disable RSS, the following command can be used from an elevated command prompt:

E:\bin>netsh int tcp set global rss = enabled

Ok.

E:\bin>netsh int tcp set global rss = disabled

Ok.

These settings are global to the operating system and affect all network adapters in the system. An administrator can enable or disable RSS for a specific network adapter by navigating to the advanced properties for the NIC in device manager. Network adapters that support RSS have an entry in the advanced properties called “Receive-Side Scaling,” which can be enabled or disabled.

## RSS Limitations

The types of packets that are received limit RSS load balancing. Load balancing on a per-connection basis is supported only for TCP. Depending on the hash type setting, other packet types such as the UDP, IPsec, IGMP, and ICMP are hashed on the source and destination IP address. For incoming packets that are not IP packets (for example, on Ethernet this would be a different EtherType than the one assigned to IPv4 or IPv6), the packets cannot be classified and are handled in a manner that is similar to non-RSS receive processing, where no hash value is set and all packets are indicated on a single CPU.

If an application is not running on the CPU on which RSS has scheduled the receive traffic to be processed, some cache optimizations may not occur. Even with this issue, application performance on RSS is still expected to be significantly better than performance with the non-RSS receive processing logic. However, in very rare conditions on nonuniform-memory-access (NUMA) systems where this issue arises, the system administrator may want to disable RSS.

Finally, RSS may need to be configured on systems where network processing is restricted to a subset of the processors in the system. Systems with large processor counts (for example 16- and 32-way processors) may not want all processors concurrently processing network traffic. In these cases, RSS should be restricted to a subset of the processors.

# Resources

The documentation on RSS is part of the Windows Driver Kit. For released products the [WDK documentation is available on MSDN](http://msdn2.microsoft.com/en-us/library/aa972908.aspx). For products that are not released yet, the WDK is available on the <http://connect.microsoft.com> Web site.

For general questions about RSS and NDIS 6.*x,* send an e-mail message to ndis6fb@microsoft.com.

Additional information can be found at:

PCI SIG specifications (including MSI-X)

<http://www.pcisig.com/home>

Toeplitz hash reference

”LFSR-based Hashing and Authentication”, Hugo Krawczyk, IBM T.J. Watson Research Center. 1998

1. MSI and MSI-X are supported in Windows Vista and Windows Server 2008. Windows Server
 2003 does not support MSI and MSI-X. [↑](#footnote-ref-2)