**Last Update:** 5 August 2018

RS5 Validation Guide

**Summary:** This document provides implementation steps to validate the dynamic spreading of host-based processing for network workloads.

Table of Contents

[Overview 3](#_Toc521265204)

[Description 3](#_Toc521265205)

[Required Hardware 3](#_Toc521265206)

[Required Software 3](#_Toc521265207)

[Test Design 4](#_Toc521265208)

[Troubleshooting and Feedback 5](#_Toc521265209)

[Prerequisites 5](#_Toc521265210)

[Install and Configure the Required Systems 5](#_Toc521265211)

[Configure Sender Firewall Rule 5](#_Toc521265212)

[Configure Receiver Firewall Rule 5](#_Toc521265213)

[Test Activities 5](#_Toc521265214)

[Activities 6](#_Toc521265215)

[Activity 1: Setup the Receiving Host and Virtual Machine 6](#_Toc521265216)

[Task 1.1 Setup the Host 6](#_Toc521265217)

[Task 1.2 Setup the Virtual Machine 7](#_Toc521265218)

[Example Receiving System Setup 7](#_Toc521265219)

[Activity 2: Baseline with Static vRSS 8](#_Toc521265220)

[Task 2.1 Configure Static vRSS on the Virtual Network Adapter 8](#_Toc521265221)

[Task 2.2: Initiate low-traffic to the Virtual Machine 9](#_Toc521265222)

[Task 2.3: Ramp throughput on the Traffic Receiver (SUT) 12](#_Toc521265223)

[Activity 3: Validate the Dynamic Algorithm 15](#_Toc521265224)

[Task 3.1: Initiate low-traffic to the Virtual Machine 15](#_Toc521265225)

[Task 3.2: Low-Throughput Switch to Dynamic 16](#_Toc521265226)

[Task 3.3: Ramp throughput on the Traffic Receiver (SUT) 19](#_Toc521265227)

[Activity 4: Quickly Maximize Throughput 21](#_Toc521265228)

[Task 4.1 Park Receive Queues 21](#_Toc521265229)

[Task 4.2 Immediate Maximum Throughput 23](#_Toc521265230)

# Overview

Software defined data centers require optimal host-based processing to provide top-class performance to tenant workloads. In Windows Server 2019, Hyper-V continues to challenge performance boundaries by dynamically spreading processing for network workloads across CPU cores for synthetic (through the vSwitch) network workloads.

In this validation guide, we will provide steps to observe and validate the dynamic spreading of processing for network workloads.

# Description

Cloud Service Providers are continually looking to identify methods to increase density on their hyper-converged or disaggregated compute workloads and network throughput is increasing to meet such demand. At the same time, tenant throughput is also continuing to climb. This high density, high throughput workload must traverse the virtual switch in to apply packet encapsulations, ACLS, SDN virtual network policies, and more. All of this leads to higher processing burdens on the host.

In the past, Virtual Machine Queue and Virtual Machine Multi-Queues enabled hardware-based packet interrupt processing. This enabled much higher throughput to individual VMs as network throughputs first reached the 10GbE mark and beyond. Unfortunately, the planning, baselining, tuning, and monitoring required for success became a large undertaking; often more than the IT Administrator intended to expend.

To address this, Windows Server 2019 improves these optimizations by dynamically spreading the processing for network workloads as required. This removes the configuration burden for administrators and ensures peak efficiency.

* Self-Tuning! Expands or coalesces across logical processors as necessary
* Dynamically reassignes queues to logical processors to maximize processing efficiency of network traffic

# Required Hardware

|  |  |
| --- | --- |
| Hardware | Description |
| Physical host system with virtualization capability and at least a 10 GbE Network adapter with available Virtual Machine Queues | This system will host a virtual machine that receives network traffic. Virtual Machine Queue (VMQ) is a required feature on the physical network adapter. To see the benefit of this feature, at least a 10 GbE adapter is also required. |
| Additional host system | This system will be used to send traffic to the virtual machine. There are no specific requirements for this adapter (virtualization is not required) however the network adapter must be capable of sending the required level of network throughput (e.g. 10 GbE or greater is required) |

# Required Software

|  |  |
| --- | --- |
| Software | Description |
| Windows Server 2019 | Latest Insider or EEAP build |
| Any network throughput generator | [NTTTCP](https://gallery.technet.microsoft.com/NTttcp-Version-528-Now-f8b12769) is used in this example but [CTSTraffic](https://github.com/Microsoft/ctsTraffic) or another traffic generator could also be used |

# Test Design

To test this feature, two physical systems are required:

**Sender** is a separate physical system. The **Receiver** or **System Under Test** (SUT) is a Windows Server 2019 virtual machine with a vmNIC.

Sender

192.168.51.11

Hyper-V Host

Receiver

Virtual Machine

192.168.51.31

# Troubleshooting and Feedback

If you encounter any challenges during your validation, please submit an issue on [GitHub](https://github.com/Microsoft/SDN/) with the [**High Performance Networking**](https://github.com/microsoft/sdn/issues?q=is%3Aissue+label%3A%22High+Performance+Networking%22) tag.

# Prerequisites

### Install and Configure the Required Systems

* Hyper-V Host System
  + Windows Server 2019 must be installed on the host
  + Hyper-V must be installed, and a Virtual Switch created. If the adapter(s) are teamed it must use Switch Embedded Teaming (SET). The use of LBFO with this feature is not supported.
  + Create one Windows Server 2019 Virtual Machine and attach the virtual switch
* Virtual Machine (Receiver)
  + [NTTTCP](https://gallery.technet.microsoft.com/NTttcp-Version-528-Now-f8b12769) should be downloaded and extracted to a folder of your choosing
  + In this article, the following IP address is assigned to this system: 192.168.51.31
* Traffic Sender: This should be a separate system that will act as the sender for the network traffic we’re generating
  + [NTTTCP](https://gallery.technet.microsoft.com/NTttcp-Version-528-Now-f8b12769) should be downloaded and extracted to a folder of your choosing
  + In this article, the following IP address is assigned to this system: 192.168.51.11

### Configure Sender Firewall Rule

NTTTCP will be blocked by the Windows (or other) firewall if a rule is not in-place to allow the traffic. Since NTTTCP can use many varied ports, you should create a firewall rule that allows the **program** to run.

For example, if NTTTCP was extracted to c:\vmSwitchTest, you could use this command to create an exception:

New-NetFirewallRule -DisplayName 'Outbound-NTTCP' -Name 'Outbound-NTTCP' `

-Program 'c:\vmSwitchTest\ntttcp.exe' `

-Direction Outbound -Action Allow

**Note:** The command above is a single command that stretches to multiple lines due to the size of the page.

### Configure Receiver Firewall Rule

NTTTCP will be blocked by the Windows (or other) firewall if a rule is not in-place to allow the traffic. Since NTTTCP can use different ports, you should create a firewall rule that allows the **program** to run.

For example, if NTTTCP was extracted to c:\vmSwitchTest, you could use this command to create an exception:

New-NetFirewallRule -DisplayName 'Inbound-NTTCP' -Name 'Inbound-NTTCP' `

-Program 'c:\vmSwitchTest\ntttcp.exe' `

-Direction Inbound -Action Allow

**Note:** The command above is a single command that stretches to multiple lines due to the size of the page.

# Test Activities

Below are the activities that we will cover in this guide:

1. Setup the Receiving Host and Virtual Machine – Open the necessary perf counters, etc.
2. Baseline with Static vRSS – First test static vRSS to create a baseline for comparison
3. Validate the Dynamic Algorithm – Observe the dynamic algorithm in action
4. Quickly Maximize Throughput – Maximize throughput as quickly as possible

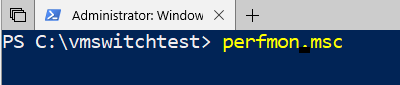
# Activities

## Activity 1: Setup the Receiving Host and Virtual Machine

In this activity, we will open and configure the necessary performance counters on the host and ready the virtual machine to receive traffic.

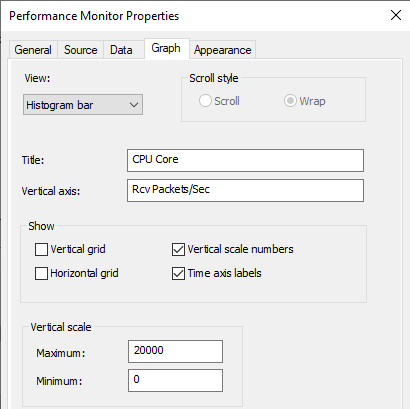
### Task 1.1 Setup the Host

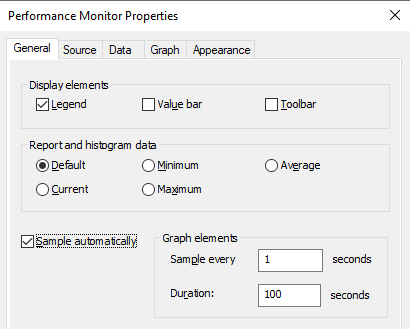
1. On the host system where the receiving VM resides, open two performance monitor windows



1. In the first performance monitor window, Click the green plus sign to add counters
2. Add the counters: **Hyper-V Virtual Switch Processor** – **Packets from External/Sec** – **All Instances**

**Note**: You may also need to modify additional properties to best display the necessary data. In this example we have modified the following properties

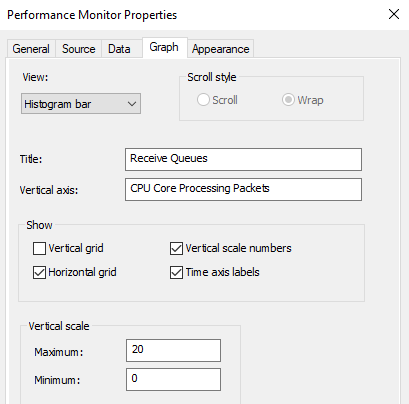


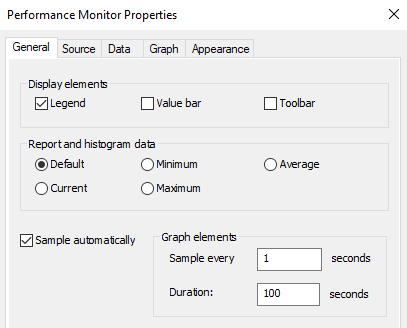


1. In the second performance monitor window, Click the green plus sign to add counters
2. Add the counters: **Hyper-V Virtual Network Adapter VRSS** – **ReceiveProcessor** – *VM01\_Network Adapter\_Entry…*

**Note:** The available entries on your system may differ from what you see in this demo as they are hardware and software configuration dependent. E.g. if your SUT is not named VM01, but VM02, you would need to add the counters specific to VM02.

**Note**: You may also need to modify additional properties to best display the necessary data. In this example we have modified the following properties





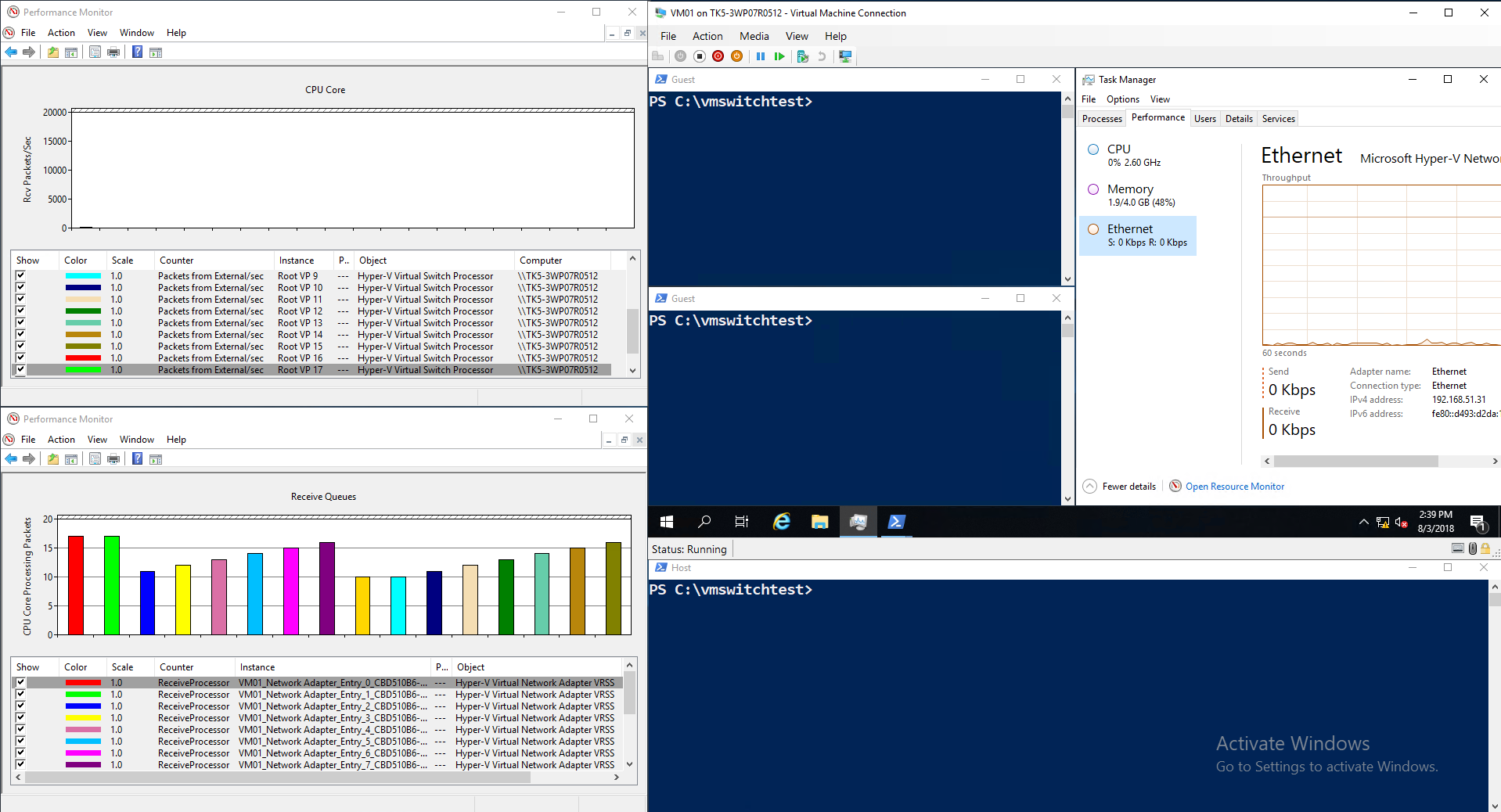
1. Next, open an elevated PowerShell prompt on the host machine
2. Finally, open a console connection to the virtual machine (System under Test or SUT)

### Task 1.2 Setup the Virtual Machine

1. In the console Windows to the System Under Test, open 2 elevated PowerShell windows
2. Next open Task Manager

### Example Receiving System Setup

At the end of Task 1.1 and 1.2, your system should look similar to this. As previously noted, the exact views may differ as they are hardware and software configuration dependent. This may include Host Processor numbers, VM names, etc.



Observe the performance counters in the bottom left-hand corner of the picture above. The receive queues are “parked” on specific cores which is to say that they have been pre-allocated and assigned to process their data on a particular CPU while they are waiting for receive traffic (they are not receiving any traffic as evidenced by the performance counter in the top-left of the screenshot).

## Activity 2: Baseline with Static vRSS

Static vRSS is an existing option available in Windows Server 2016 and prior (it is still available on Windows Server 2019). In this example, we will first set the system in Static vRSS mode, then test the system for throughput and scalability. This will give us a baseline and allow us to observe what occurs in the virtual machine when the host’s cores are overburdened.

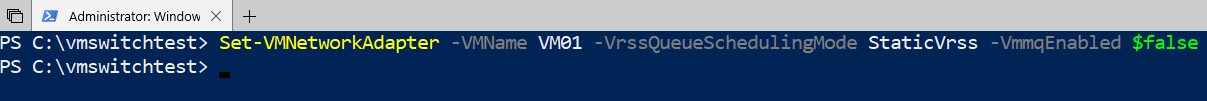
With this configuration you should see:

* **Low-throughput systems**: The host system pre-allocates VMQs across multiple processors. Once established, this correlation of queues to CPUs (indirection table) does not change. The result is an inefficient use of resources as the system utilizes more processors than required to process the incoming network data.
* **High-throughput systems**: Despite a specific processor being unable to keep up with the network data being processed on that core, it cannot move the workload to an under-utilized or available processor – correlation of queues to CPUs (indirection table) cannot change in this mode. As a result, the throughput to the virtual machine cannot be maintained.

### Task 2.1 Configure Static vRSS on the Virtual Network Adapter

In this task we’ll configure the individual virtual network adapter to use static vRSS.

From the Hyper-V host, at an elevated PowerShell prompt, run the following command:

*Set-VMNetworkAdapter -VMName* ***VM01*** *-VrssQueueSchedulingMode StaticVrss -VmmqEnabled $false*

Note: VM01 is the name of the virtual machine from the Hyper-V console

Next run the following command

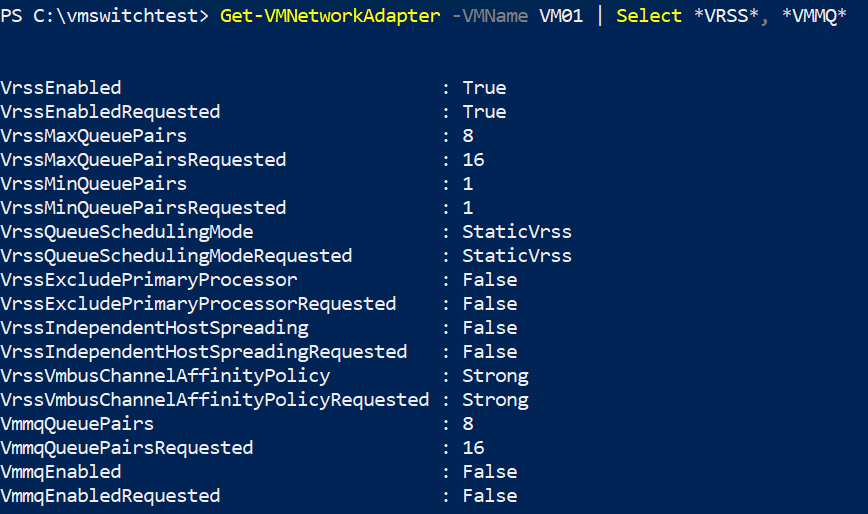
*Get-VMNetworkAdapter -VMName VM01 | Select \*VRSS\*, \*VMMQ\**

Verify that the following settings are configured as show:

**VrssEnabled** : True

**VrssQueueSchedulingMode** : StaticVrss

**VmmqEnabled** : False



**Note:** It may take a few moments for the values to change as intended.For example, you may find thatthe settings report:

VrssEnabled: **False**

VrssEnabledRequested: **True**

If you find that **vRSSEnabled** remains **False** despite **vRSSEnabledRequested** at **True**, run the **Get-VMNetworkAdapter** command above again. If the issue persists, disconnect and reconnect the vmNIC from the virtual machine.

### Task 2.2: Initiate low-traffic to the Virtual Machine

In this task, we’ll initiate a low amount of traffic to the receiving VM.

1. On the **Sender**, open a PowerShell window and navigate to the folder that has NTTTCP.
2. Next, run NTTTCP to begin sending traffic to the SUT. The goal is to limit the traffic to what can be processed by 1 or 2 processor cores. For example, on our test system

**.\NTTTCP -s -a 8 -m 64,\*,192.168.51.31 -t 9999 -thr 10000**

In the example above:

**-s** defines this is a sender system

**-a** sets the outstanding IOs to 8

**-m** 64 thread

**-t 9999** time in seconds to send traffic

**-thr 10000** Throttle the send data with throughput specified in KB/s for each thread

1. On the **Receiver** (Guest VM), open a PowerShell window and navigate to the folder that has NTTTCP.
2. Next, run NTTTCP to begin receiving traffic on the SUT. The goal is to limit the traffic to what can be processed by 1 or 2 processor cores. For example, on our test system

**.\NTTTCP -r -a 8 -m 64,\*,192.168.51.31 -t 9999**

In the example above:

**-r** defines this is a receiver system

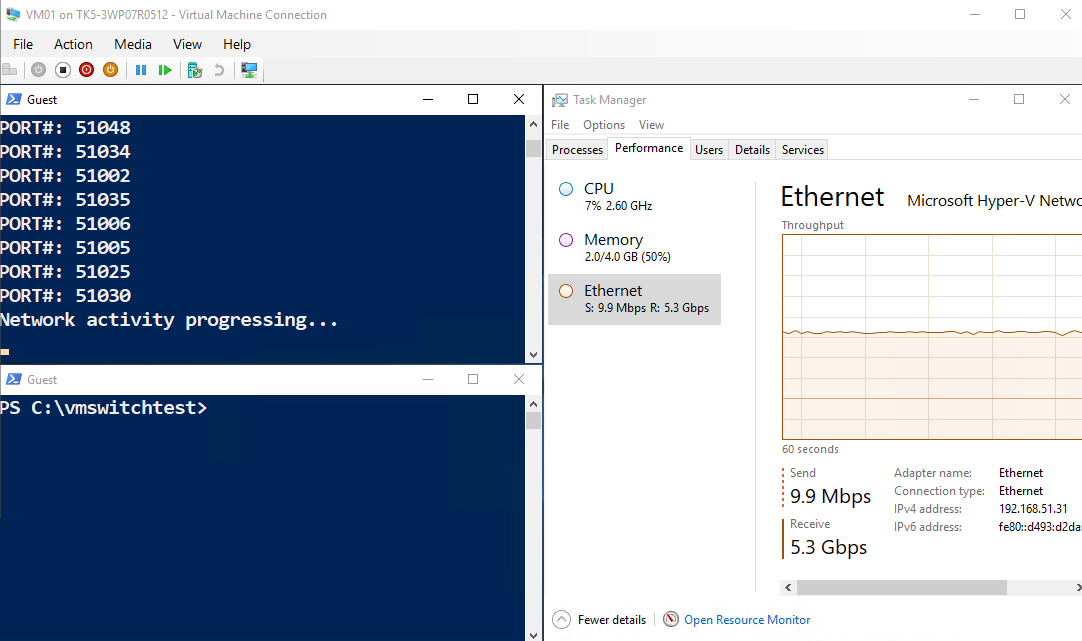
**-a** sets the outstanding IOs to 8

**-m** 64 thread

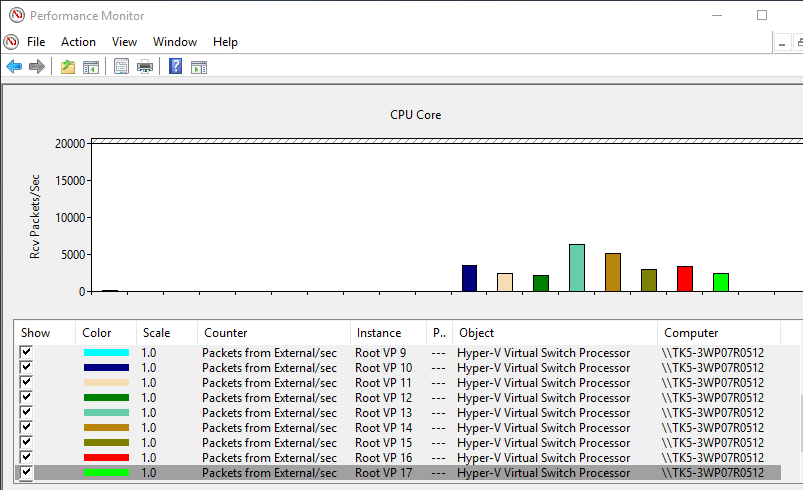
**-t 9999** time in seconds to send traffic

**Note:** The goal of this test is to limit the network traffic such that the system **could** process the network data on 1-2 cores. You may need to change throttling (**thr** parameter on the sender) to manipulate this.

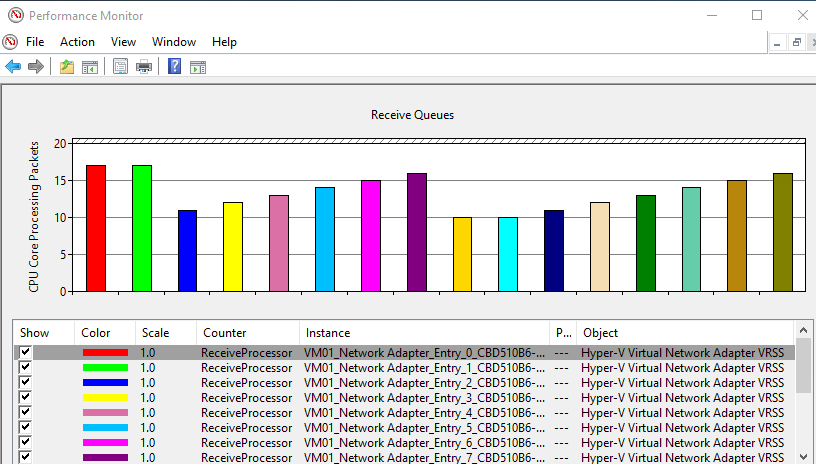
In the SUT (Guest VM), wait until the NTTTCP command changes to **Network Activity Progressing…** Once this occurs, the receive traffic should increase as shown in the screenshot below.



As you can see above, the VM is receiving 5.3 Gbps of network throughput. This traffic is being processed by multiple processors (shown below) despite the cores appearing to be able to process more data.



The receive queue assignments (indirection table) have not changed. The receive queues remain spread (in the picture below, 2 queues per CPU, for a total of 8 CPUs and across multiple CPUs despite needing only 1 – 2 cores to process the existing workload. This is an inefficient use of processing power as there is a tax paid to distribute packets to another processor.



**Note on Task Manager:** The natural inclination is the open task manager to view the network processing per CPU. Task manager has several deficiencies in these scenarios. For example, task manager cannot distinguish between receive and send traffic nor can it highlight the Base VMQ in this scenario.

We recommend that you rely on the performance counters for these tests as this can lead to inaccurate conclusions of the data.

### Task 2.3: Ramp throughput on the Traffic Receiver (SUT)

In this task, we’ll now increase the traffic on the receiving VM.

1. On the **Sender**, open a PowerShell window and navigate to the folder that has NTTTCP.
2. Next, run NTTTCP to begin sending traffic to the SUT. The goal is to increase the traffic to the max for the VM with the existing configuration. For example, on our test system

**.\NTTTCP -s -a 8 -m 64,\*,192.168.51.31 -t 9999**

In the example above:

**-s** defines this is a sender system

**-a** sets the outstanding IOs to 8

**-m** 64 thread

**-t 9999** time in seconds to send traffic

**-thr 10000** this parameter is removed to remove the throttling

1. On the **Receiver** (Guest VM), open a PowerShell window and navigate to the folder that has NTTTCP.
2. Next, run NTTTCP to begin receiving traffic on the SUT. The goal is to limit the traffic to what can be processed by 1 or 2 processor cores. For example, on our test system

**.\NTTTCP -r -a 8 -m 64,\*,192.168.51.31 -t 9999**

In the example above:

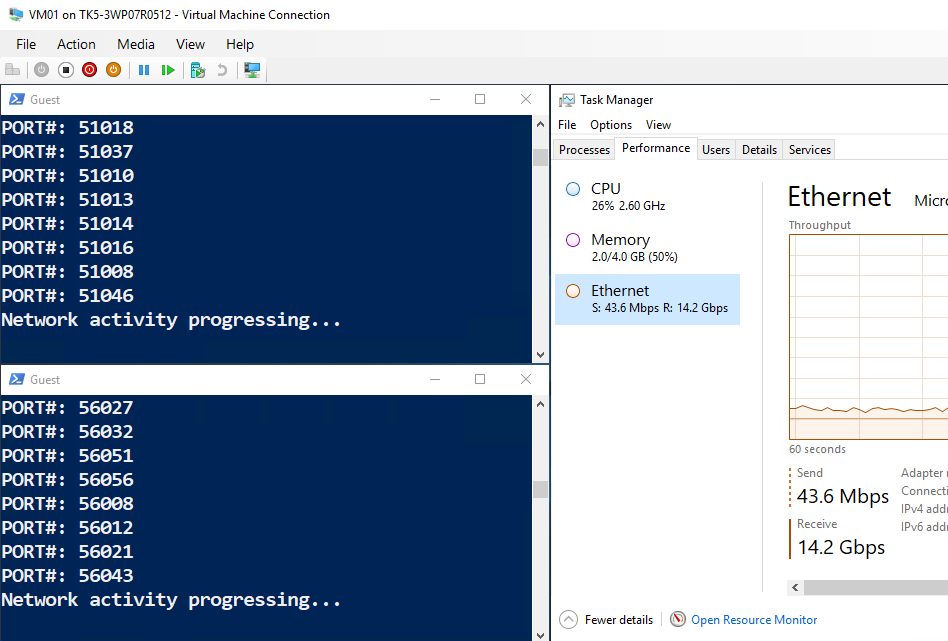
**-r** defines this is a receiver system

**-a** sets the outstanding IOs to 8

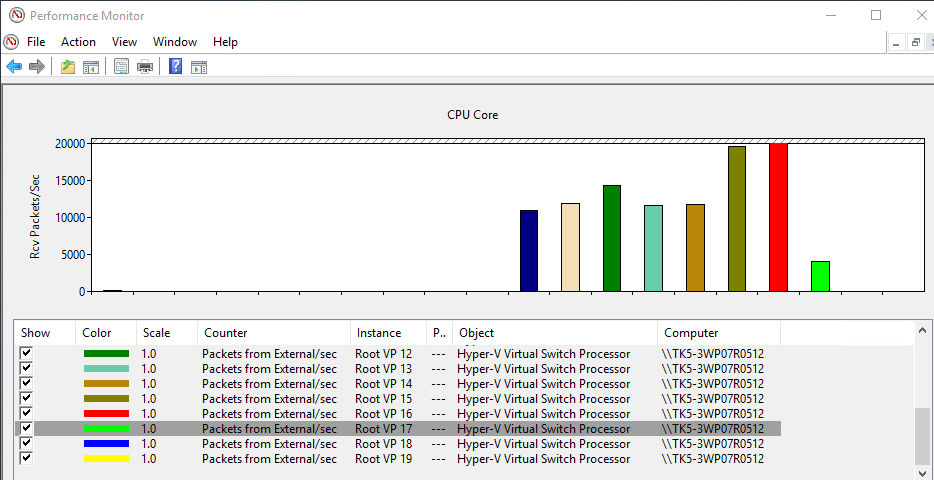
**-m** 64 thread

**-t 9999** time in seconds to send traffic

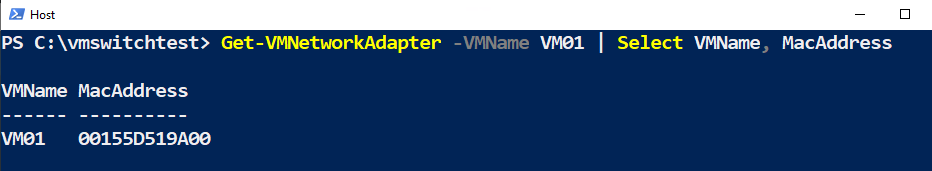
In the SUT (Guest VM), wait until the NTTTCP command changes to **Network Activity Progressing…** Once this occurs, the receive traffic should increase as shown in the screenshot below.



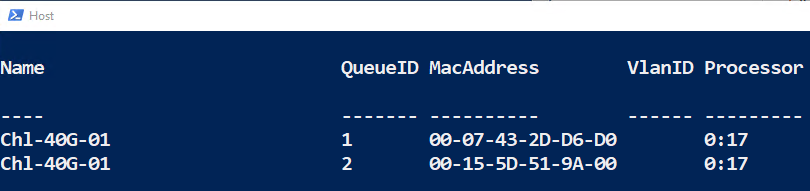
As you can see above, the VM is receiving 14.2 Gbps of network throughput. This traffic is being processed by same processors as before. Some of the cores have peaked and this is limiting our throughput to the virtual machine despite the NICs being capable of higher throughputs (these are 40 Gbps NICs in our scenario).



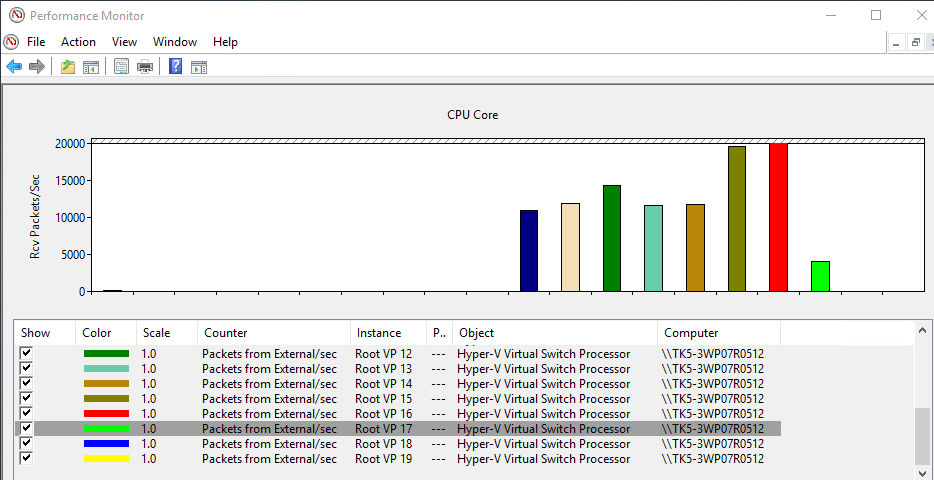
Next, run the following command on the elevated PowerShell window on the host to determine the MAC Address of the vmNIC. In the screenshot below, the vmNIC’s MAC Address is **00155D519A00**.



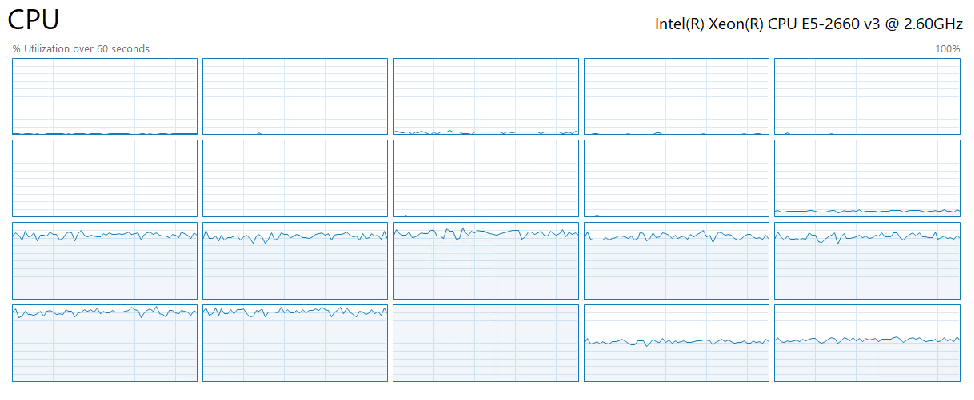
Now run the following command to determine the BaseVMQ for the vmNIC. Correlate the MAC Address from the previous command with the entries displayed here. Note the entry for Processor. In this example, the Processor for the BaseVMQ at MAC Address **00-15-5D-51-9A-00** is Processor 17.



As you can see in the picture below, Processor 17 (green bar to the far right) is not processing many packets itself. As the processor for the BaseVMQ, its job is to receive the packets and distribute them to other processors for processing.



By looking at Task Manager on the host, you can see that processor 17 is completely consumed. This limits the throughput to the virtual machine because the host is using one of its processors to distribute packets rather than process packets itself. In addition, the other cores all have available processing cycles that are unused.



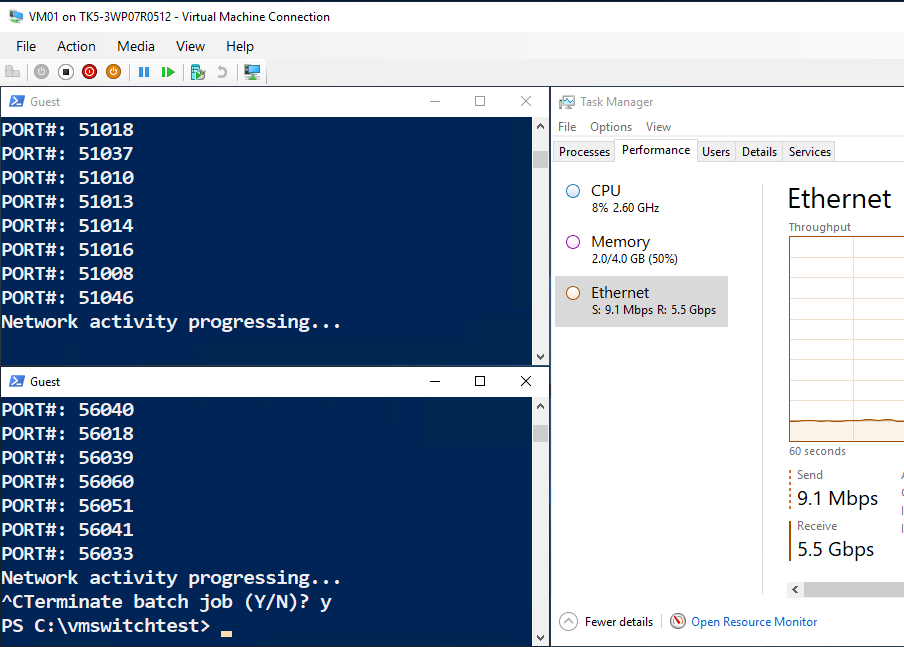
Next let’s see how the dynamic algorithm introduced in Windows Server 2019 help solve some of these problems.

## Activity 3: Validate the Dynamic Algorithm

### Task 3.1: Initiate low-traffic to the Virtual Machine

In this task, well reset the traffic flow to match that of Task 2.2.

On the Guest VM, stop the second transfer as shown below

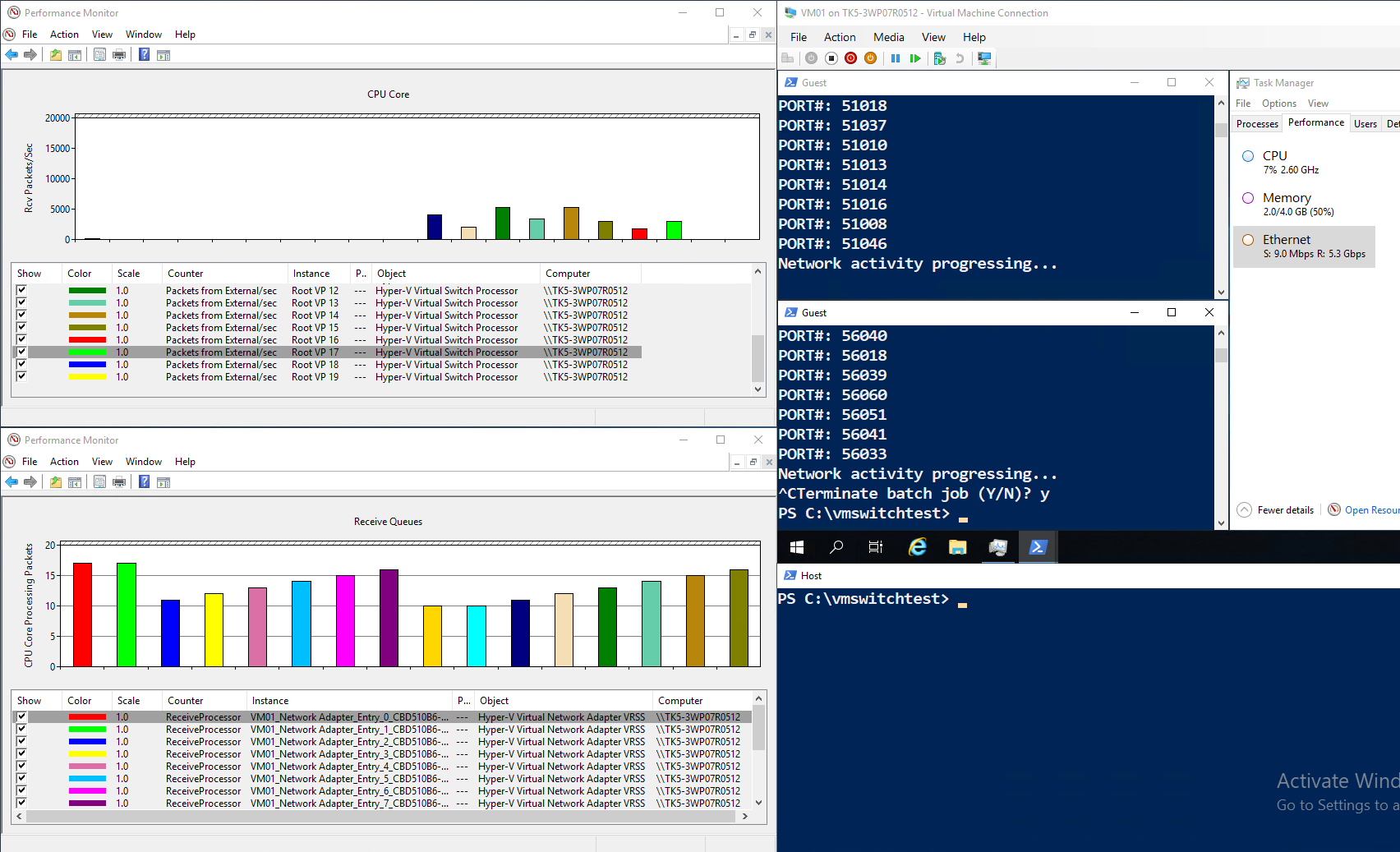


**Note**: The transfer and throughput return to approximately 5.5 Gbps

### Task 3.2: Switch Low-Throughput system to Dynamic Algorithm

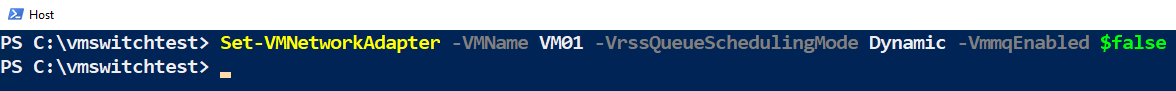
In this task, we’ll switch to the dynamic algorithm and observe how it coalesces the processing of network data to as few processors as required.

Prior to beginning this task, note the existing setup of the performance counters. With 5.5 Gbps still coming into the VM, the packets are still being received on several processors. This is because the StaticVRSS algorithm has placed the Queues and will not move them once they are set.

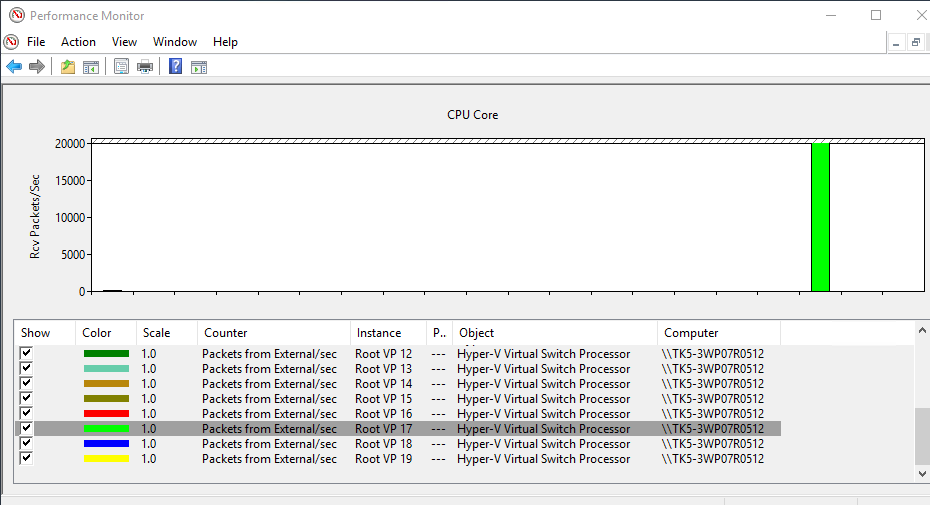


In the elevated PowerShell prompt on the host, run the following command to switch to the dynamic algorithm.

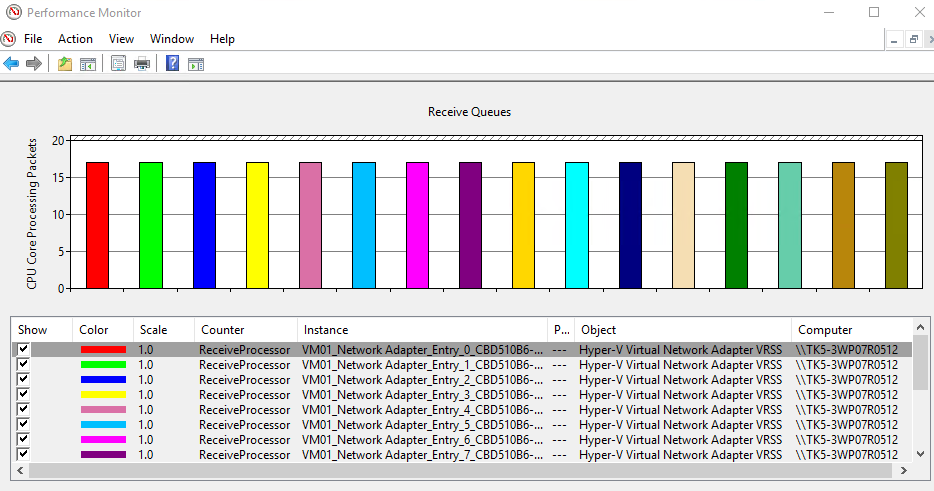
Set-VMNetworkAdapter -VMName VM01 -**VrssQueueSchedulingMode** **Dynamic** -VmmqEnabled $false



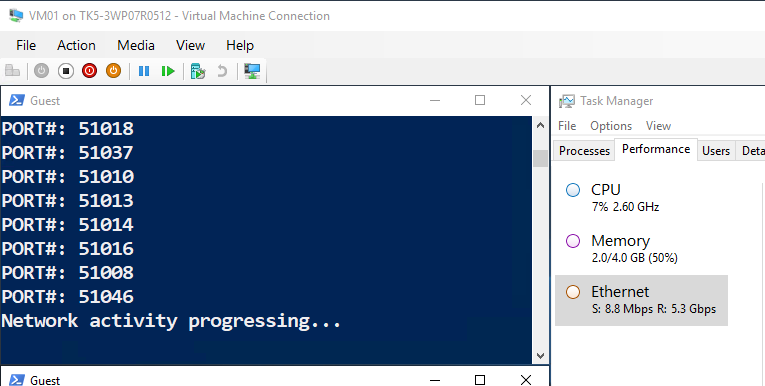
In this picture below, you can see that all receive packets are now being processed on Processor 17, the BaseVMQ we identified earlier.



Each receive queue is now processing its data on CPU 17. This is far more efficient that engaging multiple CPUs.



Most importantly, the throughput to the VM is not affected. The virtual machine data is stable while the host processing has been optimized.



**Note:** It may take some time to balance the work load as this depends on a few factors. For example, the specific NTTTCP command used or the number of queue moves in the recent past may affect the speed at which queue moves occur. If you don’t see the expected result as shown above,

* Wait approximately 60 seconds (this may take a bit more or less time). The delay is to prevent “thrashing” whereby a system would be moving queues around to frequently.
* Alternatively, review the exact NTTTCP command used; it may need to be tweaked for your specific NIC throughput or system (CPU’s)

### Task 3.3: Ramp throughput on the Traffic Receiver (SUT)

In this task, we’ll now increase the traffic on the receiving VM. The goal of this is to see the dynamic algorithm automatically expand to additional processors as needed while autotuning the queues involved to maximize processing efficiency.

1. On the **Sender**, open a PowerShell window and navigate to the folder that has NTTTCP.
2. Next, run NTTTCP to begin sending traffic to the SUT. The goal is to increase the traffic to the max for the VM with the existing configuration. For example, on our test system

**.\NTTTCP -s -a 8 -m 64,\*,192.168.51.31 -t 9999 -thr 10000**

In the example above:

**-s** defines this is a sender system

**-a** sets the outstanding IOs to 8

**-m** 64 thread

**-t 9999** time in seconds to send traffic

**-thr 10000** throttle the send data with throughput specified in KB/s for each thread

1. On the **Receiver** (Guest VM), open a PowerShell window and navigate to the folder that has NTTTCP.
2. Next, run NTTTCP to begin receiving traffic on the SUT. The goal is to limit the traffic to what can be processed by 1 or 2 processor cores. For example, on our test system

**.\NTTTCP -r -a 8 -m 64,\*,192.168.51.31 -t 9999**

In the example above:

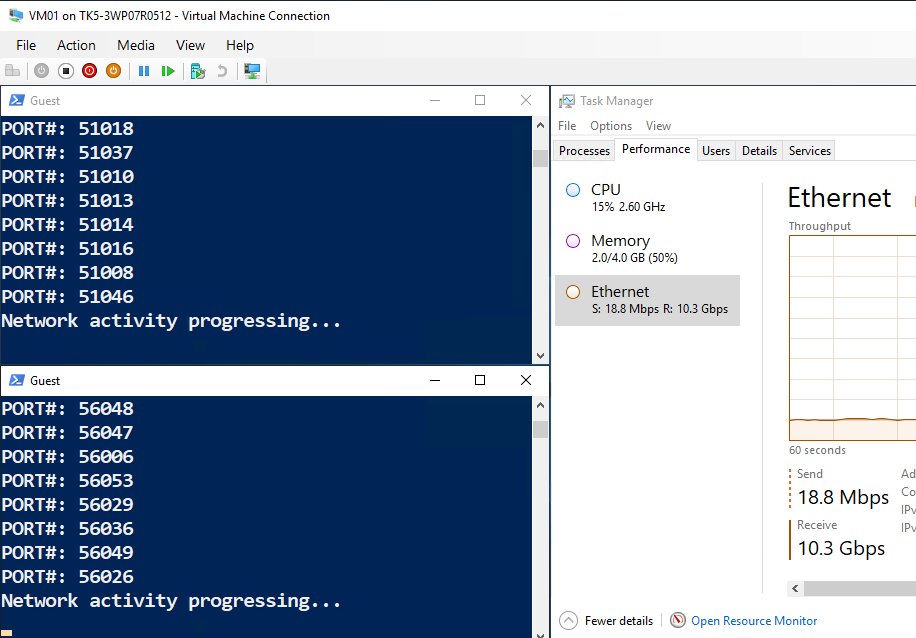
**-r** defines this is a receiver system

**-a** sets the outstanding IOs to 8

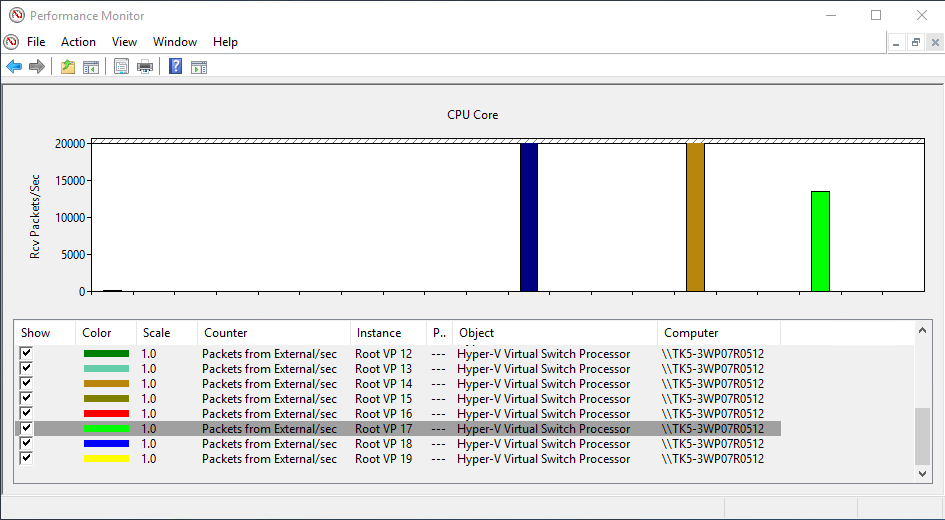
**-m** 64 thread

**-t 9999** time in seconds to send traffic

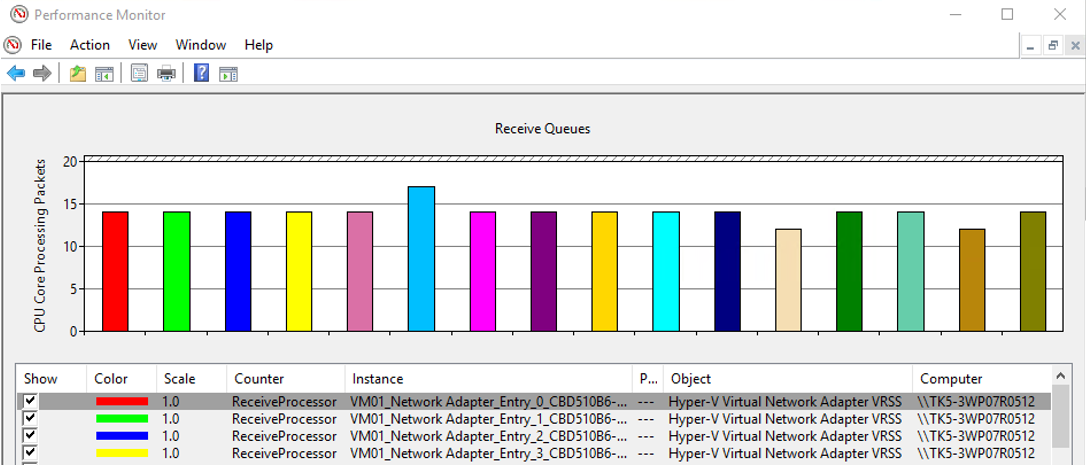
In the SUT (Guest VM), wait until the NTTTCP command changes to **Network Activity Progressing…** Once this occurs, the receive traffic should increase as shown in the screenshot below. Note the throughput has increased to 10.3 Gbps



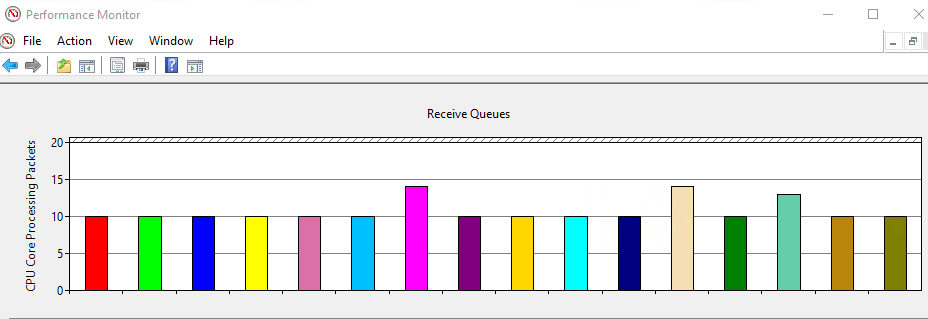
Two additional receive queues have been engaged in addition to the BaseVMQ.



The three receive queues have spread to three CPUs to process the data (the height of the bar below indicates the CPU engaged by the receive queues).



Overtime, you may see the dynamic algorithm continue to balance the queues to continually maximize efficiency

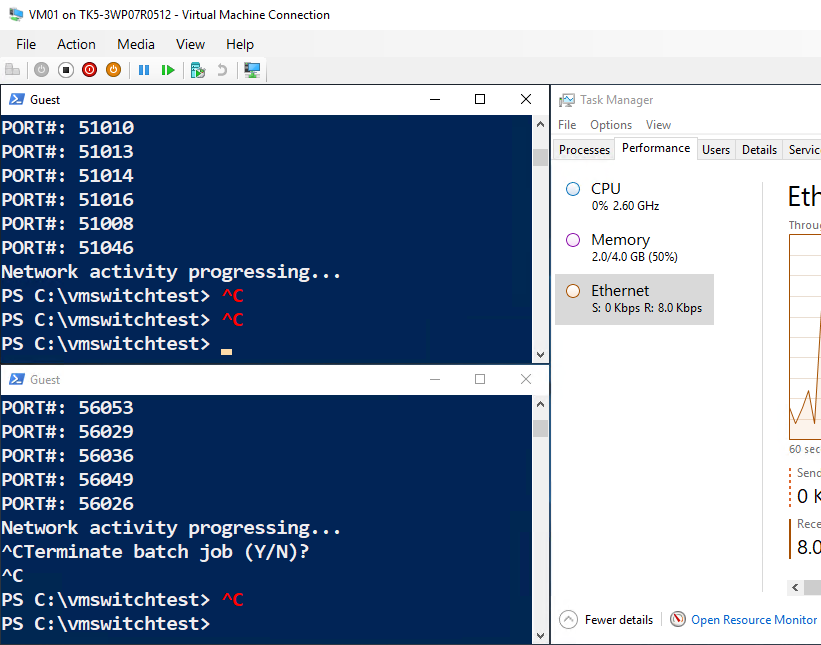


## Activity 4: Quickly Maximize Throughput

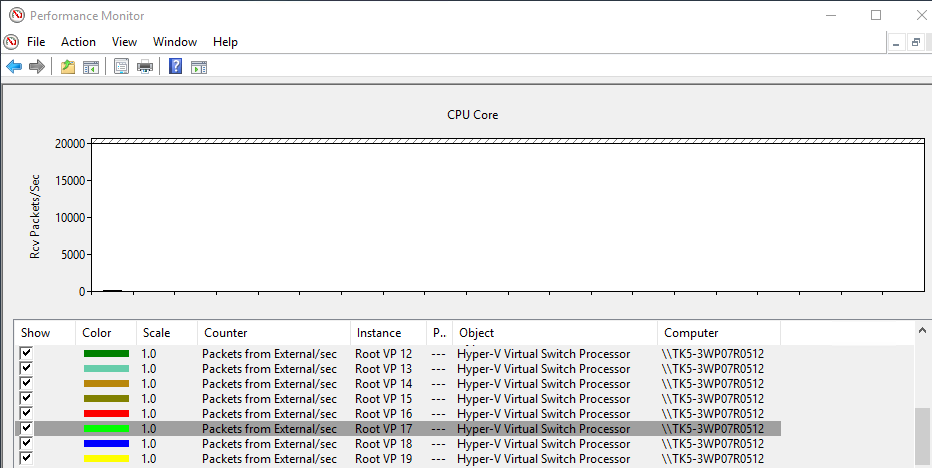
In this activity we will observe how the receiving system can immediately expand to maximum throughput. As you can imagine if, all queues were sitting on a single CPU, they could not be engaged simultaneously – 1 to 2 receive queues would typically consume an entire processor.

### Task 4.1 Park Receive Queues

1. Cancel the NTTTCP workloads

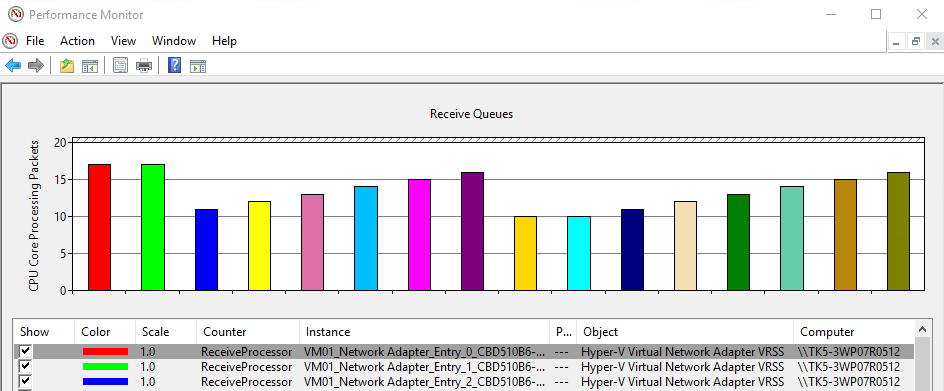


1. Note that the receive queues are no longer receiving network traffic



1. As a result, the receive queues “park” on separate CPUs by the dynamic algorithm. This allows the network traffic to ramp very quickly in the event of a throughput spike.

In the example below, two receive queues are parked per CPU core.



### Task 4.2 Immediate Maximum Throughput

In this task, we’ll now maximize the traffic to an idle receiving VM. The goal of this is to observe the benefit of the parked receive queues and how the system quickly reaches maximum throughput. They dynamic algorithm automatically expands or coalesces the CPUs as necessary to reach the maximum throughput while ensuring maximum host processing efficiency.

1. On the **Sender**, open a PowerShell window and navigate to the folder that has NTTTCP.
2. Next, run NTTTCP to begin sending traffic to the SUT. The goal is to increase the traffic to the max for the VM with the existing configuration. For example, on our test system

**.\NTTTCP -s -a 8 -m 64,\*,192.168.51.31 -t 9999**

In the example above:

**-s** defines this is a sender system

**-a** sets the outstanding IOs to 8

**-m** 64 thread

**-t 9999** time in seconds to send traffic

1. On the **Receiver** (Guest VM), open a PowerShell window and navigate to the folder that has NTTTCP.
2. Next, run NTTTCP to begin receiving traffic on the SUT. The goal is to limit the traffic to what can be processed by 1 or 2 processor cores. For example, on our test system

**.\NTTTCP -r -a 8 -m 64,\*,192.168.51.31 -t 9999**

In the example above:

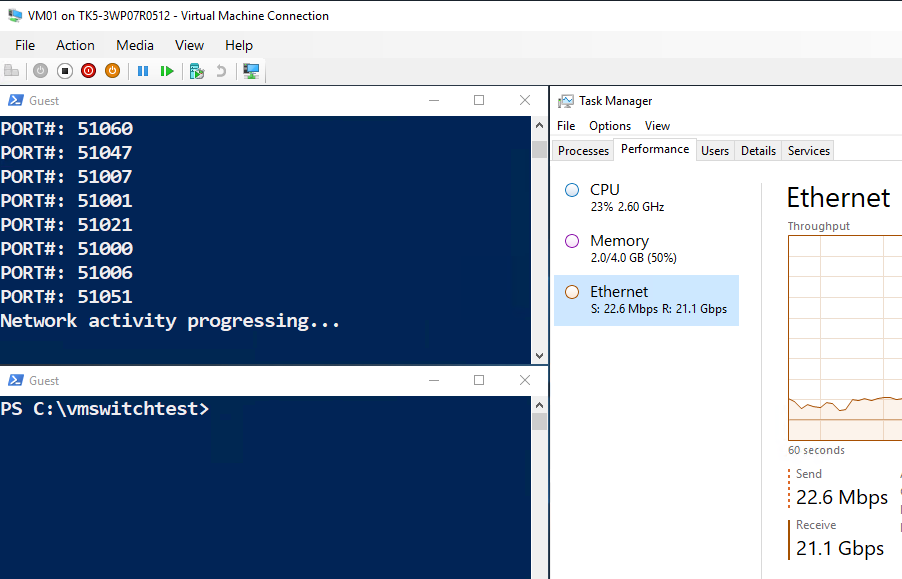
**-r** defines this is a receiver system

**-a** sets the outstanding IOs to 8

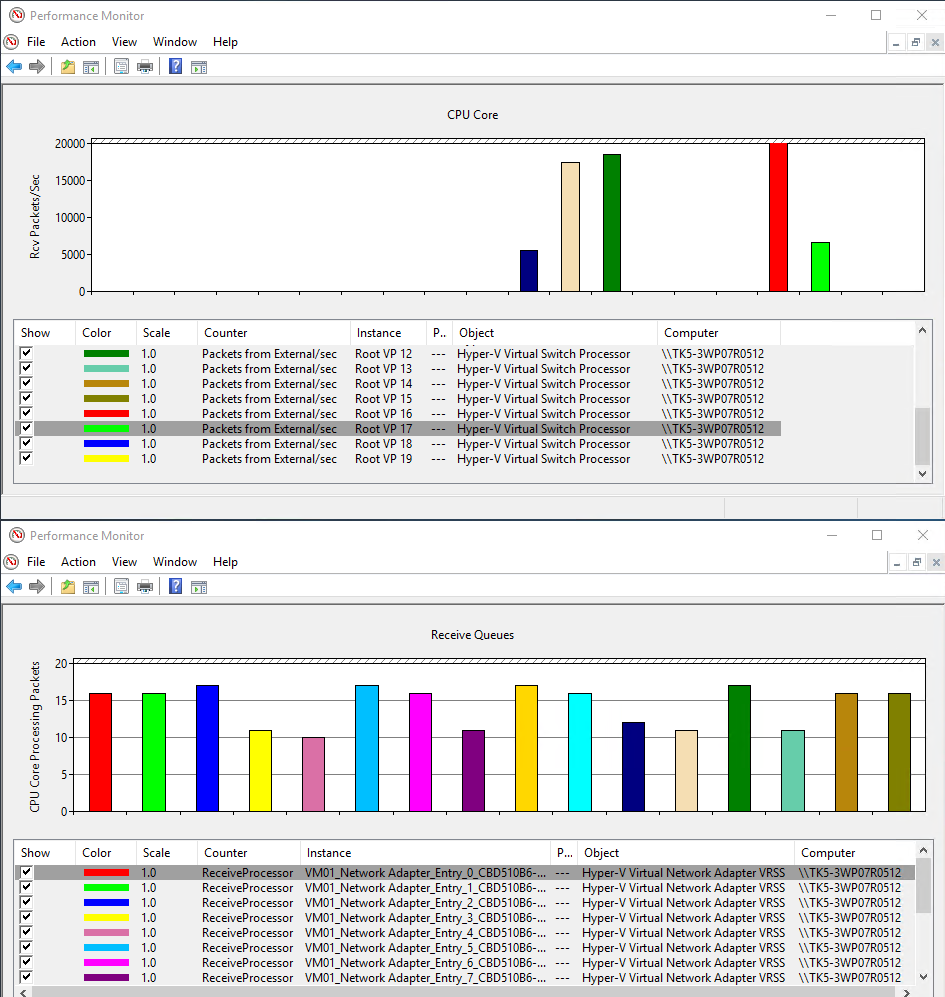
**-m** 64 thread

**-t 9999** time in seconds to send traffic

In the SUT (Guest VM), wait until the NTTTCP command changes to **Network Activity Progressing…** Once this occurs, the receive traffic should increase as shown in the screenshot below. Note the throughput has increased to 21.1 Gbps



As a result of the network throughput, you might see that the performance counters are not updated immediately. Wait a few minutes and you should see the performance counters return.



The important aspect to note is that the network throughput to the virtual machine immediately expands to the maximum.

**Note:** Depending on the speed of your NICs, you may not be able to reach maximum throughput for that device. For example, the test system used with this guide, have 40 Gbps NICs however the virtual machine can obtain no more than ~20 Gbps. This is because we disabled VMMQ earlier in this guide.

VMMQ is a hardware offload that allows the NIC to place the receive packets directly onto the appropriate processor. With VRSS alone, the BaseVMQ consumes an entire processor and must distribute the receive packets to the appropriate processor. With VMMQ, this distribution of packets is no longer required, and an additional processor is available for receiving packets.

In the future, we will expand this guide to include testing of VMMQ with the dynamic algorithm.