Network Performance Improvement for FreeBSD Guest on Hyper-V

Dexuan Cui <decui@microsoft.com>
Yanmin Qiao (Sephe) <yaqia@microsoft.com>
Hongjiang Zhang <honzhan@microsoft.com>

Open Source Technology Center, Microsoft

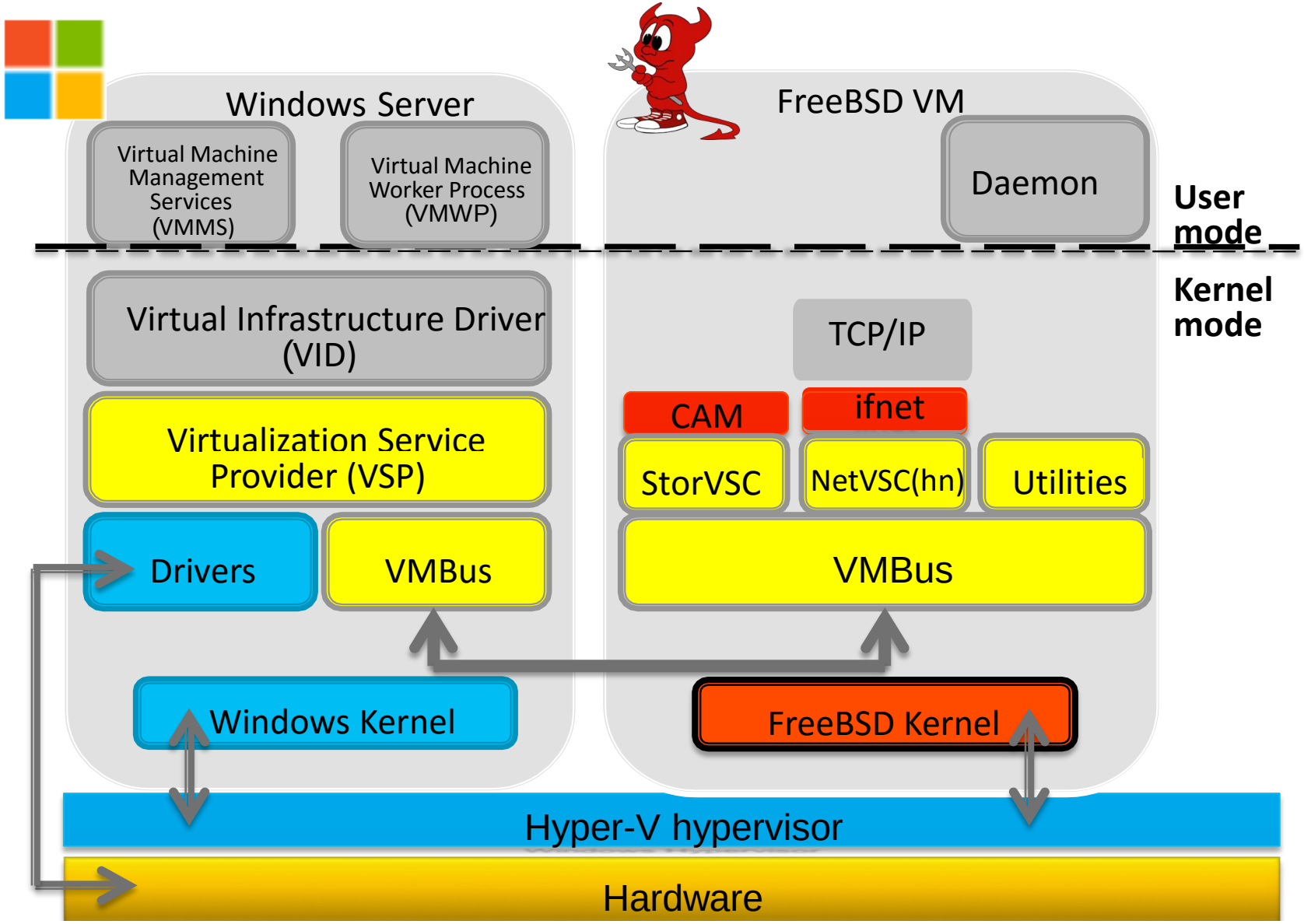
Outline

- Introduction to Integration Service for FreeBSD (BIS)
- Our work on Hyper-V network driver (NetVSC or hn(4))
  - VMBus multi-channel
  - Checksum offload
  - TCP Segmentation Offload
  - Large Receive Offload
  - Virtual Receive Side Scaling
- Performance achievement

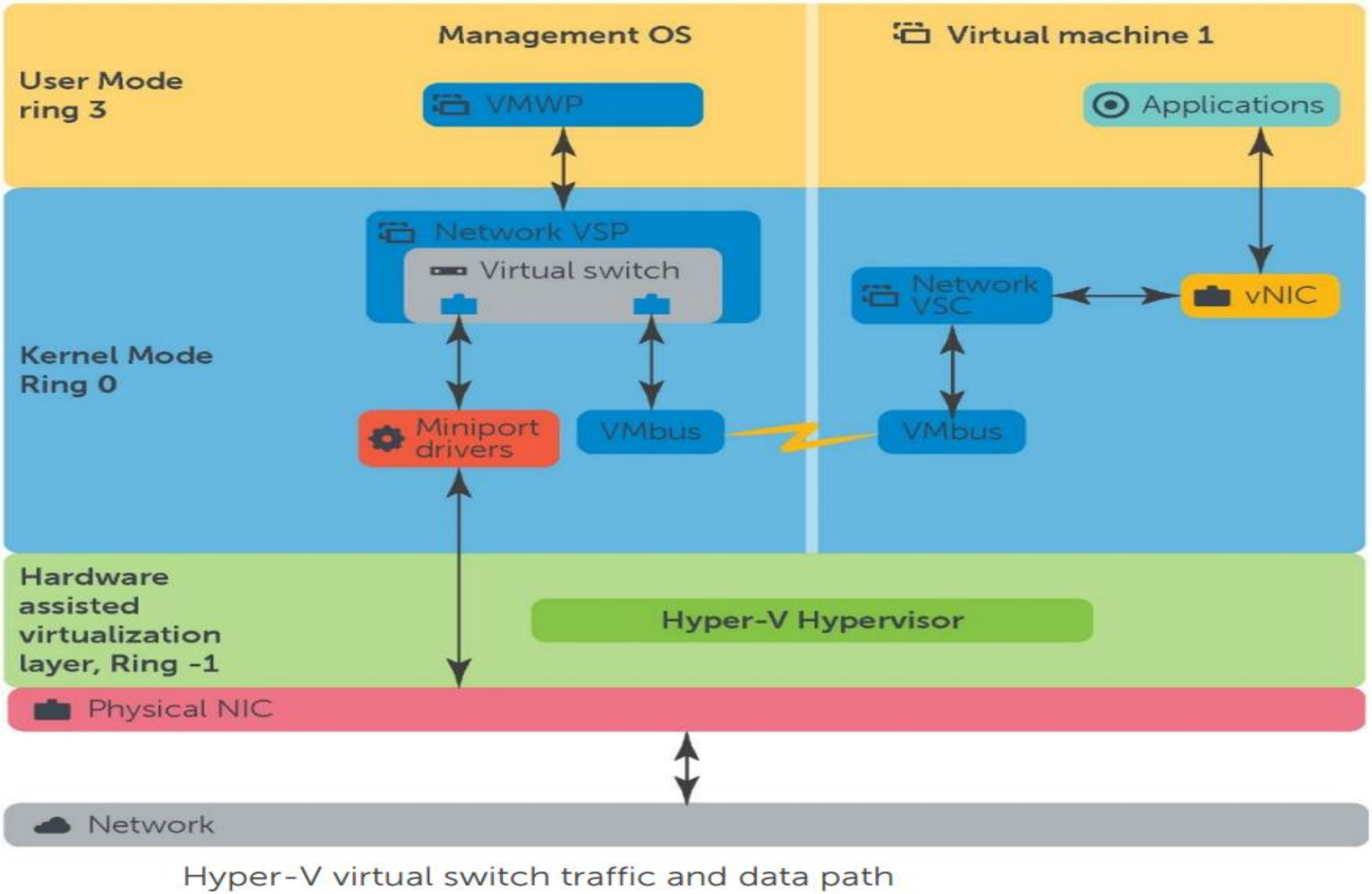
Integration Service for FreeBSD (BIS)

- Hyper-V presents synthetic devices to the guest OS
  - Synthetic devices seen by the guest OS are the same, regardless of the real hardware under Hyper-V
- Guest OS needs drivers for these synthetic devices
  - Just like an OS needs drivers for devices it sees when running on real hardware
- Integration Services == the drivers for the Hyper-V synthetic devices
  - They run in the guest OS so follow the device driver model for that guest OS
  - Also include some user-space daemons that interact with the drivers

Integration Service for FreeBSD - High Level Architecture



Networking data path in Hyper-V



Hyper-V virtual switch traffic and data path

The picture is from here.

Integration Service for FreeBSD - Evolution

- Ports available for FreeBSD 8.4, 9.1, 9.2 and 9.3.
- FreeBSD 10 has built-in BIS drivers.

FreeBSD 10, 10.1

- Lacking core support to enable I/O performance
- Supported on a “best effort” basis by Microsoft Customer Support

FreeBSD 10.2

- Enhance **core** functionality
  - **VMBus multi-channel**
  - CARP**, CSUM offload, TSO**
  - Storage sub-channel & Scatter/Gather List
  - KVP driver and daemon

FreeBSD 10.3, 11-CURRENT

- Enable **network** based FreeBSD workloads on Hyper-V & Azure
  - Enhance networking stability – 10.3
  - **LRO – 10.3**
  - **vRSS – 11-CURRENT**

NetVSC performance enhancement

- VMBus Multi-channel (10.2)
- Checksum Offload (10.2)
- TCP Segmentation Offload (10.2)
- Large Receive Offload (10.3)
- Virtual Receive Side Scaling (11-CURRENT)

- Test Environment
  - Windows Server 2012 R2 host: 32GB memory, 16 cores (2 sockets) with HT disabled (Intel Xeon CPU E5-2650 v2 @2.6GHz), **Intel 10G 2P X520 NIC (82599)**.
  - FreeBSD guest: 8 vCPUs, 4GB memory.
  - Netperf is used.

VMBus multi-channel on SMP guest(10.2)

Principle — VMBus Channel

- Class ID and Instance ID: identify a device.
  A pair of send/recv ring buffers (i.e. for NIC device: 1 Tx queue and 1 Rx queue)
  hypercall-based signaling (every channel has a vCPU bound to it).

Improvement -1 — Bind different channels to different vCPUs – 10.2

- In 10.1, all channels were bound to vCPU0 even on recent Hyper-V.

Improvement -2 — Enable multiple channels for performance-critical devices

- 1 NIC device can have multiple channels.
- This paves the way for the later vRSS support in 11-CURRENT.

CheckSum Offload (10.2)

Principle

Offload the csum calculation to the host.

- On **send**: the upper layer passes down mbufs with CSUM_IP/TCP/UDP flags; NetVSC hands over them to the backend VSP driver with the flags
- On **recv**: NetVSP hands over packets with the flags to NetVSC; NetVSC pushes them to upper layer.

Improvement

Throughput improved by ~200Mbps

- Send and Recv: 2 Gbps -> 2.2 Gbps

Other Problem

Guest interrupt handler: 100% utilization of single CPU.

- Mainly because of protocol processing
  process incoming TCP segments or ACKs, wake up process (if necessary)
  send ACKs or send pending outgoing TCP segments.

TCP Segmentation Offload (10.2)

Principle — Offload the segmentation work to Hyper-V host

- It saves lots of CPU cycles on the send side.
- On send, the upper layer passes down mbuf chains (up to 64KB)
- NetVSC hands them over to the host with the TSO RNDIS header (RNDIS == Remote NDIS)

Improvement — The **Send** throughput increases significantly.

- 2.2 Gbps → 4.5~7 Gbps
- Interrupt handler uses 60~80% of single CPU.
- TSO is crucial for send performance, especially in guest.

Other Problem — Receive side?

Large Receive Offload (10.3)

Principle — Aggregation

- Aggregate multi incoming TCP segments or ACKs from single stream into one mbuf chain
- Effectively reduce Recv protocol processing overhead
- **Significantly reduce ACKs rate**

Improvement — **Recv** throughput increases significantly.

- 2.2 Gbps → 4.5~7.5 Gbps
  Recv side: interrupt handler uses only 20~40% of single CPU.

Other Problem

- It's not always good to aggregate as many packets as possible
- Throughput instability with many concurrent connections
- Small throughput jitter

Large Receive Offload (10.3) - 1

Problem — Good to aggregate as many packets as possible?

- Pure-ACKs received: should be timely passed up to the upper layer.
  Data segments: aggregating too much can delay sending ACKs to the peer.

Solution — Enhance the LRO API

- Allow the driver to configure the per-connection limits of data length and the ACK count.
- By default, at most we aggregate 25*MTU data(for 1 queue) or 2 ACKs.

Improvement — Both Recv and Send throughputs increase again!

- Recv:
  - 4.5~7.5Gbps to 8~9.1Gbps (1 queue)
  - Interrupt handler uses 40~60% of single CPU.
- Send:
  - 4.5~7Gbps to 8~9.1Gbps (1 queue)
  - Interrupt handler uses 40~50% of single CPU (it was 60~80% with TSO only): receiving less ACKs.

Large Receive Offload (10.3) - 2

Problem Throughput instability with many concurrent connections

- Especially obvious for >64 connections per Rx ring.

Solution Enhance the LRO API

- The default 8 LRO entries per-Rx-ring are not enough.
- NetVSC allocates 128 entries by default.
  - tcp_lro_init_args()

Large Receive Offload (10.3) - 3

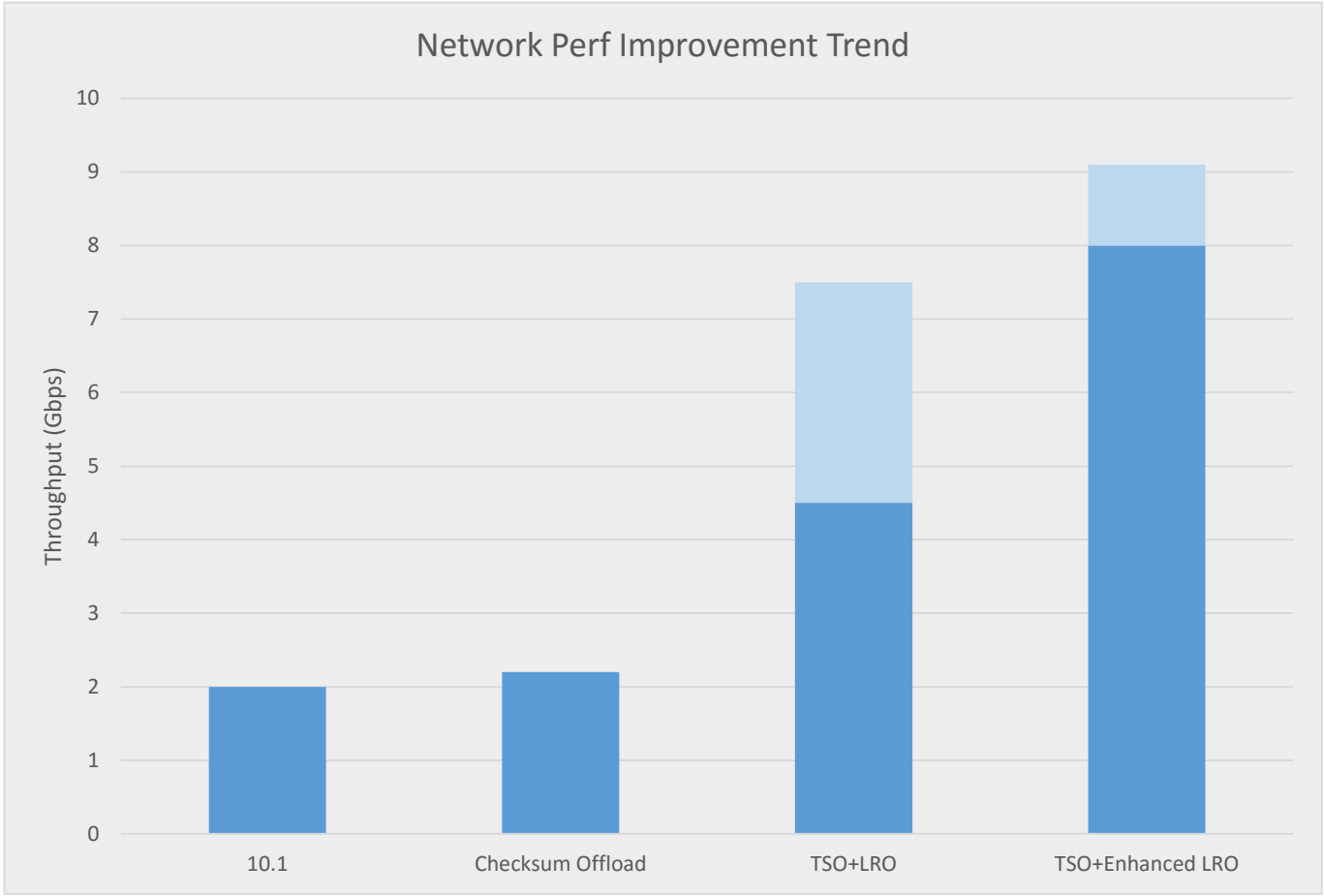
Problem Small throughput jitter.

- LRO internal data structure is not efficient for entry removal & lookup

Solution Optimize LRO internal data structure

- Fast entry removal: SLIST -> LIST
- Speed up LRO-entry lookup using hash table (not committed in 11-CURRENT yet)

Recap



How about the performance in 40Gb environment?

vRSS (11-CURRENT)

Principle

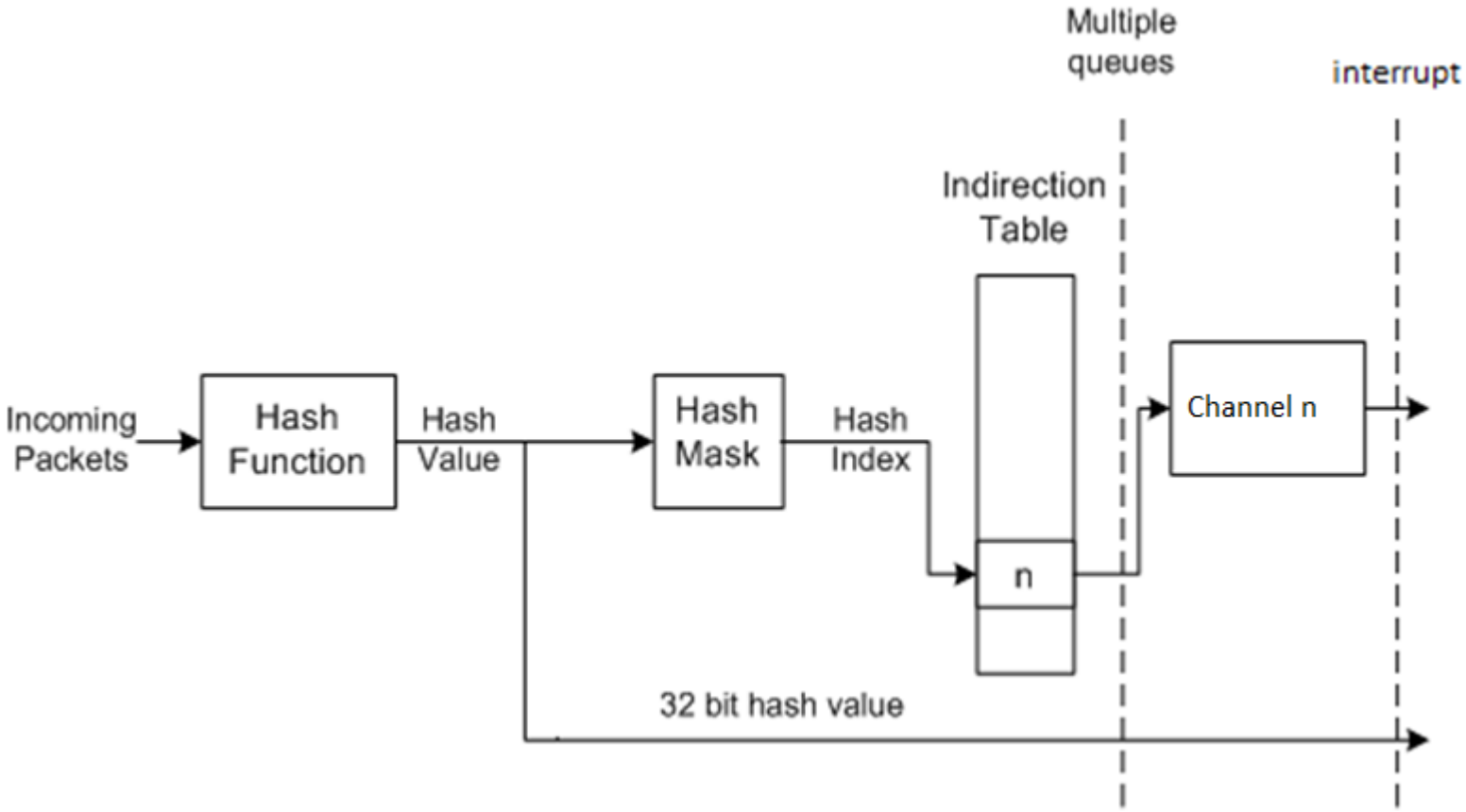
Virtual Receive Side Scaling

- Enable the host to efficiently distribute traffic to different guest vCPUs
- Multiple queues: each has its own bound vCPU.
- The same connection is handle by the same queue.
  - Reduce lock contention
  - Keep things more hot in CPU cache
- It’s crucial for really high-speed 40Gb NIC.

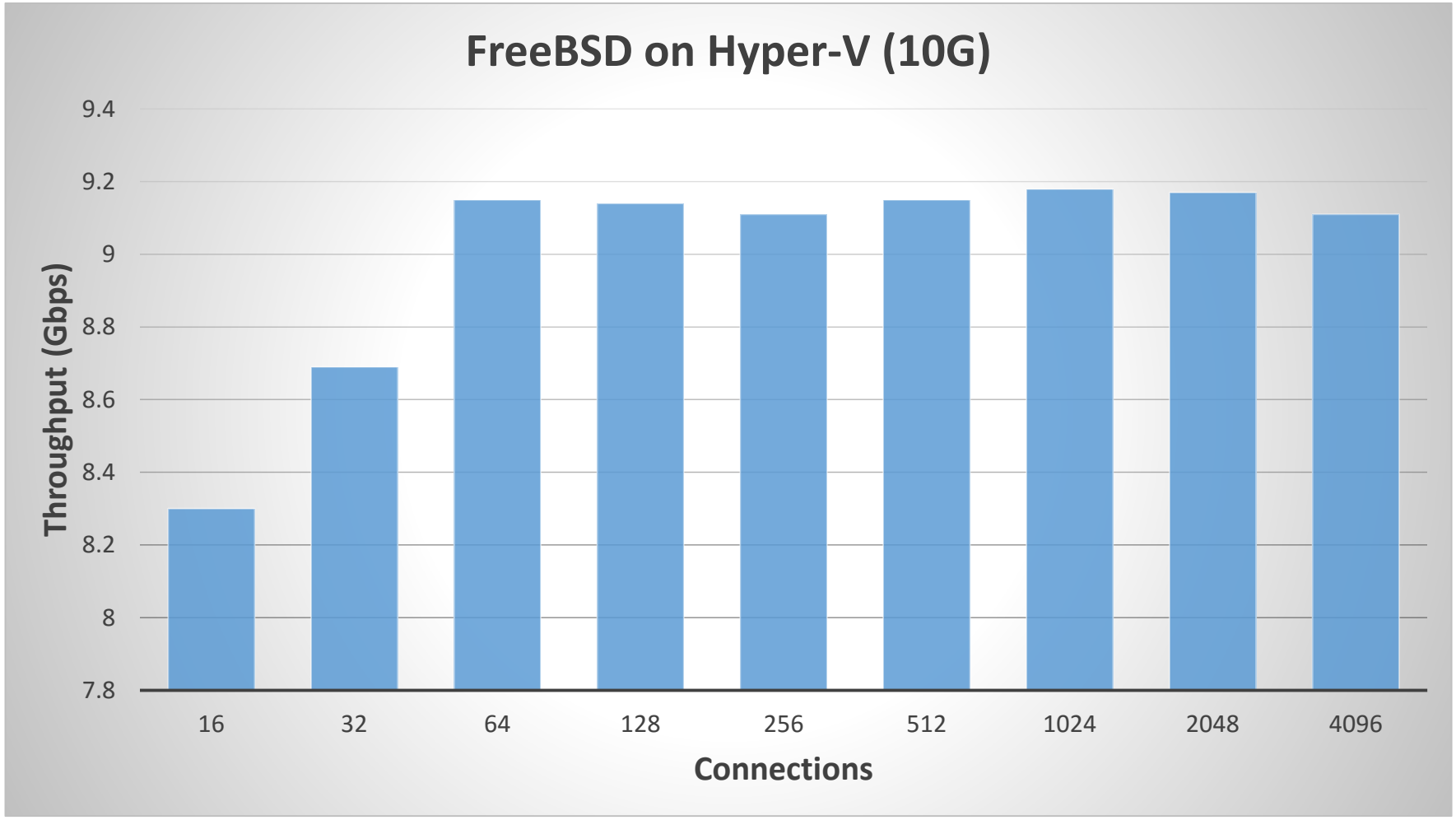
How we integrate vRSS in NetVSC

- On init:
  - Ask the host to create multiple channels (e.g. 8 vCPUS → 8 channels).
    - A channel consists of 1 TX queue & 1 RX queue.
  - Register the hash key/type/function & indirection table to the host.
    - Indirection table: hash values → channels
- On recv: we pass the hash value as flow_id, and the hash type, to the upper layer.
- On send: the same flow_id is used to choose the same channel.

How we integrate vRSS in NetVSC

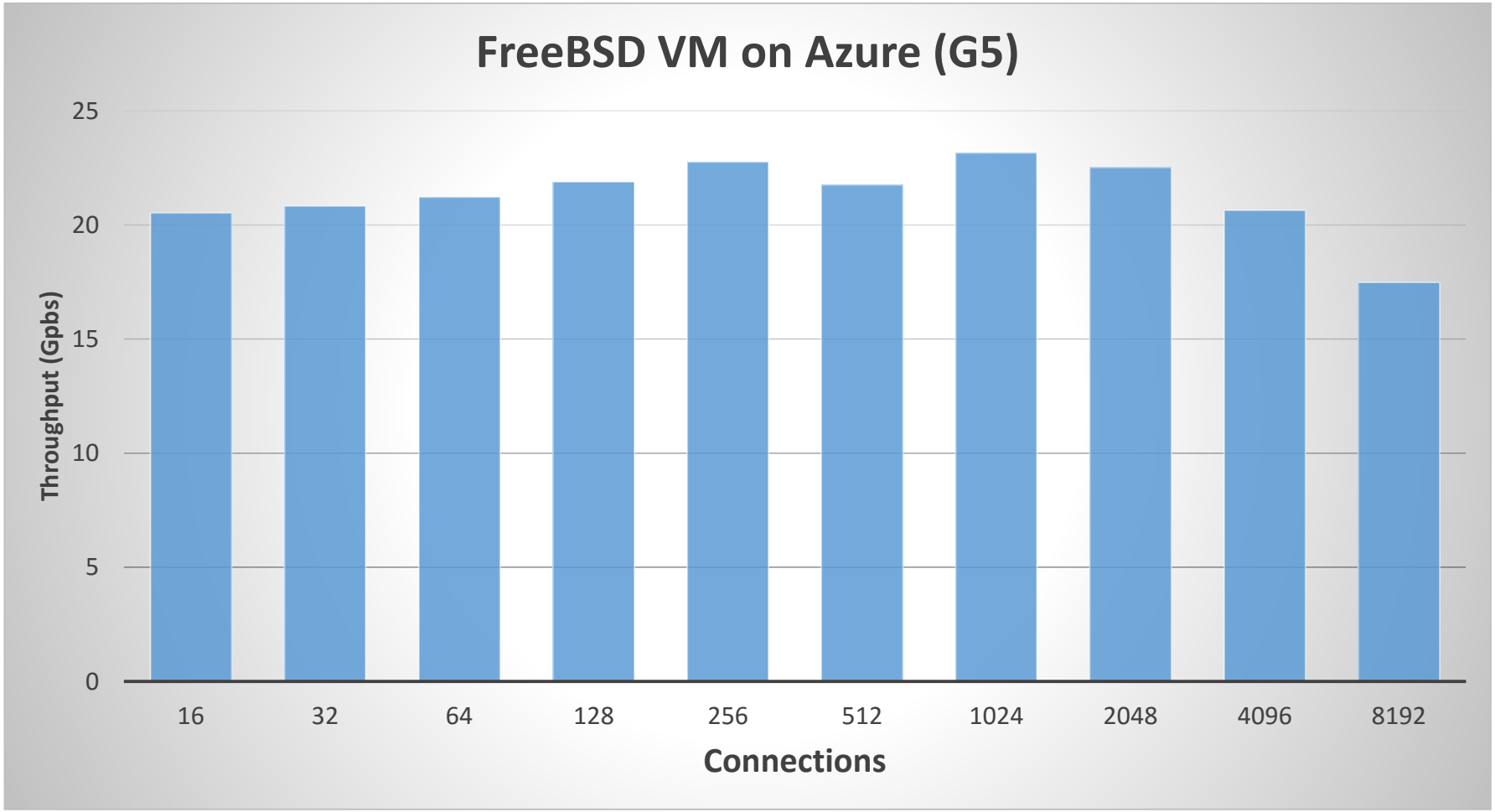


Performance data with 10Gb NIC on local Hyper-V VM



Disclaimer: this is not official performance data from Microsoft. The data could be different due to different test tools, environment, etc.

Performance data with 40Gb NIC on Azure G5 size VM

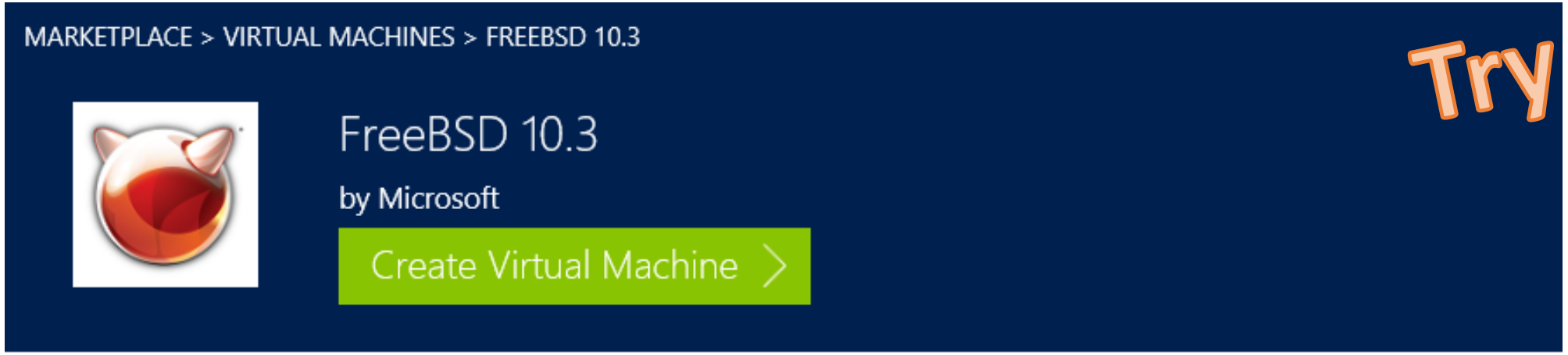


Disclaimer: this is not official performance data from Microsoft. The data could be different due to different test tools, environment, etc.

To-Do list

- Investigate how hypervisor scheduling affects performance (NUMA?)
- Analyze the throughput & latency with >8K concurrent connections
- Integrate Hyper-V network driver with FreeBSD RSS framework
- More profiling on UDP and packet forwarding
- Try netmap with Hyper-V network driver
- SR-IOV? DPDK?
- And more…

FreeBSD is available on Azure Marketplace.



Try It !

FreeBSD 10.3 for Microsoft Azure provided by Microsoft. FreeBSD is an advanced computer operating system used to power modern servers, desktops and embedded platforms.

The FreeBSD Logo and the mark FreeBSD are registered trademarks of The FreeBSD Foundation and are used by Microsoft with the permission of The FreeBSD Foundation.

Thank you

Support email bsdic@microsoft.com