Reducing the Overhead of Network Virtualization: Software and Hardware Solutions

Akihiko Odaki
akihiko.odaki@daynix.com Daynix Computing Ltd.

LINUX PLUMBERS CONFERENCE | Vienna, Austria
Sept. 18-20, 2024

Network Virtualization

Use cases:

- KVM
- Containers

Open Problems:

- Software: receive steering
- Hardware: SR-IOV

Software Network Virtualization

- **tuntap**
  - Virtual L2/L3 interface for the userspace
- **vhost_net**
  - Accelerates the interaction with tuntap for KVM
  - Linux implements the whole data path of the virtio interface
    - **Reduces the number of context switches to the userspace for KVM**
  - The userspace implements the control path
    - Complexity away from the kernel

Receive steering

Problem 1: multiple hardware threads (SMP/SMT)

- Distribute received packets to hardware threads
- Keep data locality to minimize communication among hardware threads
  - Inter-Processor Interrupt (IPI)
  - Cache

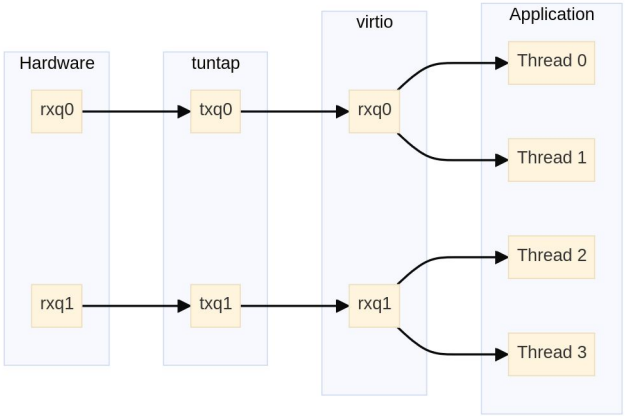
Solution 1: **receive steering**

- Distribute packets based on **flow**:
  - IP addresses
  - TCP/UDP ports

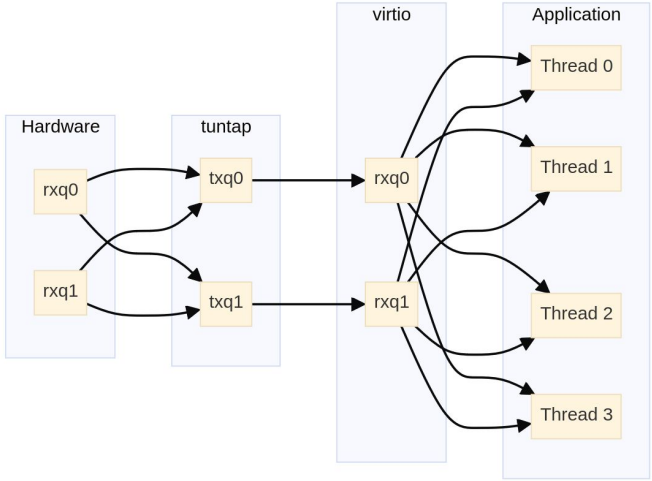
Receive steering consistency problem

Problem 2: **the whole network stack must perform consistent steering**

Ideal:



Worst case:



Receive steering consistency problem

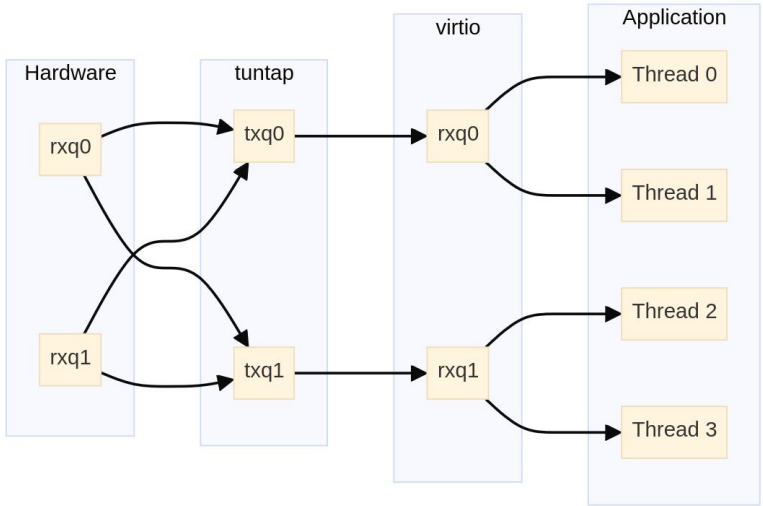
Problem 2: **the whole network stack must perform consistent steering**

- Hardware: Receive Side Scaling (RSS)
  - Old: part of Microsoft's Network Device Interface Specification (NDIS) 6.0, which is part of Windows Vista
  - Uses Toeplitz hash function
- Socket: Receive Packet Steering (RPS)
  - Uses SipHash since 5.4: faster and more secure
  - Enables receive flow steering (RFS)
    Pass packets to the hardware thread where the application resides

automq: bi-directional flow tracking

Solution 2a: [automq in tuntap]

- Procedure:
  - Remember rxq index used for a flow
  - Use the index to determine txq for the reversed flow
- Pro: **requires no negotiation**
- Con: **stateful**
  - Max. 4096 flows expiring at 3 Hz
  - No effect on new connections and connectionless protocols
- Con: **reduced hash entropy**
  - Ignores the direction of flow



VIRTIO RSS

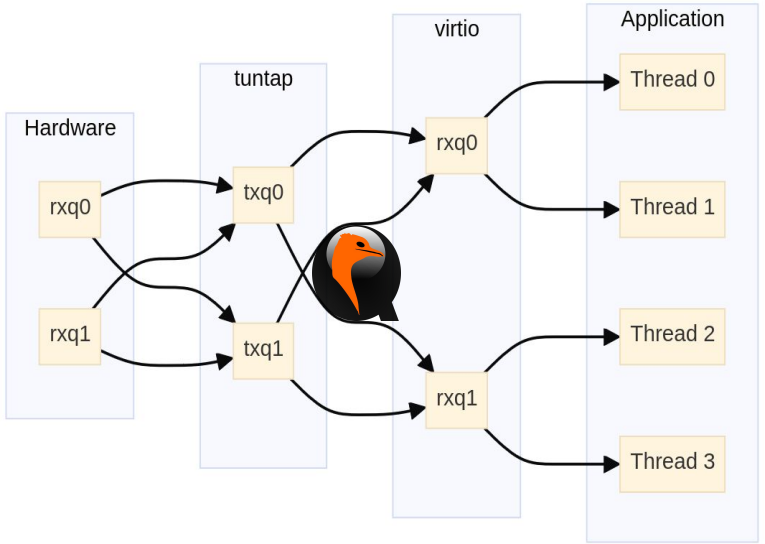
Solution 2b: **Let the guest negotiate the use of RSS**

- Pro: stateless
- Pro: more configurability for guest
- Con: unable to deal with guest thread migration
  - RSS does not know which guest threading handles a flow

2 Implementations: In-QEMU and eBPF RSS

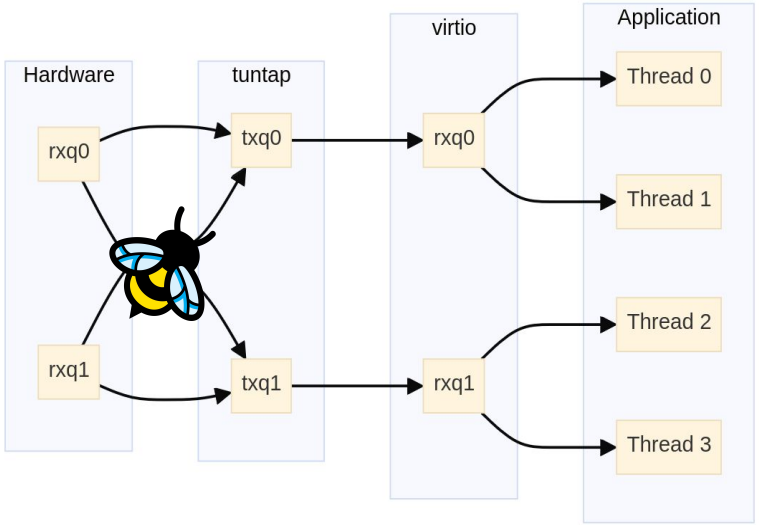
VIRTIO RSS in QEMU

- **Reference implementation**
- **Little performance benefit**: QEMU needs to exchange packets among hardware threads
- **Incompatible with vhost_net**: In-QEMU RSS requires the userspace data path



VIRTIO RSS with eBPF steering program

- Shipped with QEMU
- Attach an *eBPF steering program* to tuntap
- **Real performance benefit**
- **Compatible with vhost**
- Requires a privilege
  [Delegate program loading to libvirt]



Hash reporting

Solution 2c: **Pass computed hashes to guest through virtio**

- Pro: saves hash calculations
- Pro: consistent hash values by definition
- Does not obsolete automq/RSS
  - The mapping between hash values and queues still needs to be negotiated
  - Lacks application-specific configuration
  - Windows requires to enable RSS and hash reporting at the same time
- RFC patches for Linux and QEMU

Source of Hash Reporting

- automq
  - L4 hash provided by hardware if available
    - Better chance of achieving consistency between hardware and guest?
  - Software-computed symmetric hash otherwise
- eBPF steering program (used for RSS)
  - **The computed hash is unavailable to the kernel**
  - Relies on “context rewrite”
    - Tedious interface definition
    - Entered feature freeze around 2022
  - “kfuncs” as alternative to context rewrite are not UAPIs
    - Other relevant APIs (KVM and vhost_net) are UAPIs.

RSS in kernel

tuntap: add ioctl for RSS

- Hash values will be available in the kernel
- UAPI
- Bonus: requires no privilege

eBPF is still useful for:

- Evaluating new receive steering algorithms
- Application-specifics

Future of receive steering

- Inner header hash for encapsulated packets
- Flow filter for accelerated RFS
- RSS context for more flexibility
- Even expose eBPF to guest?
  - Needs to mitigate security concerns
    - Restrict programmability
    - Loading eBPF programs with an out-of-band privilege operation

Hardware Network Virtualization

Software network virtualization:

- Incurs overheads
- Requires complicated optimization

Let’s offload it to hardware!

PCI Single Root I/O Virtualization (SR-IOV)

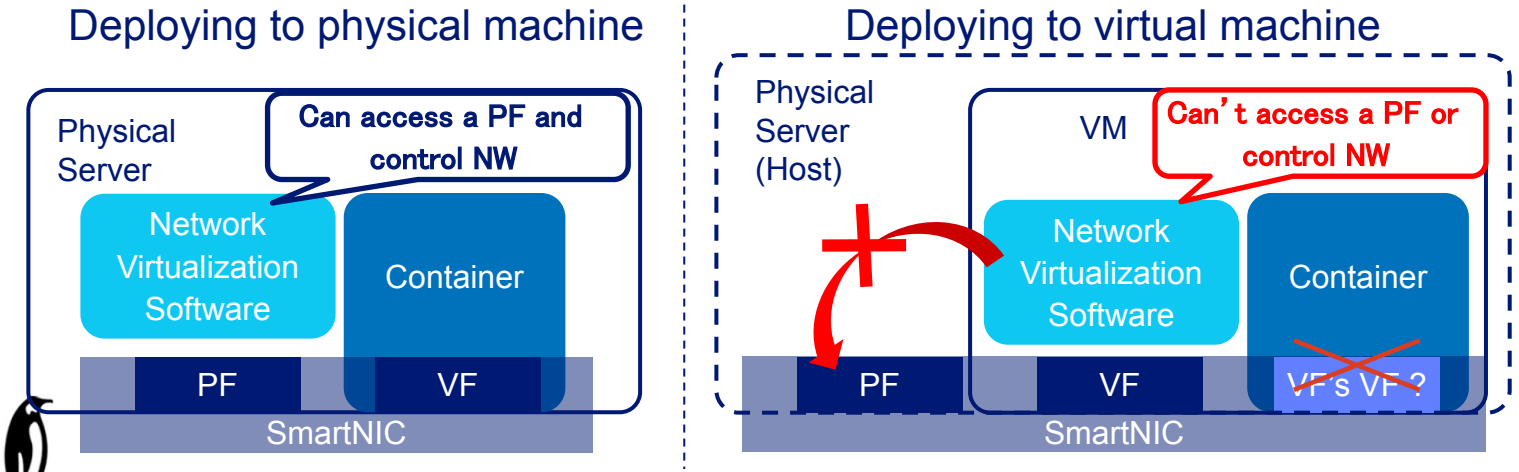
- Standardized in 2007
- Expose *virtual functions (VFs)* configured by the host to guests
  - The host configures packet switching in the networking context

Problem solved?

Container in VM

Reminder: **containers also need network virtualization**

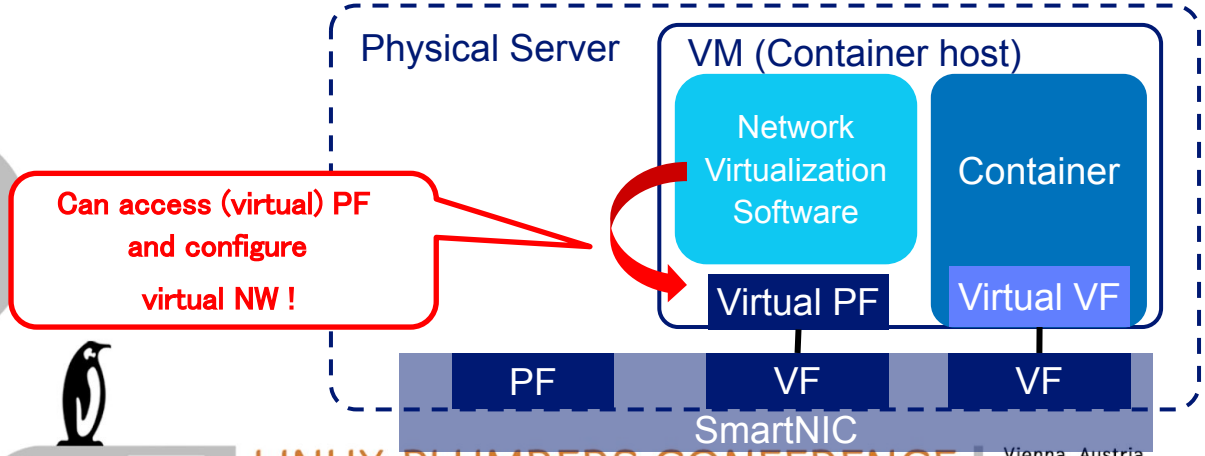
- Container in VM results in nested virtualization in terms of networking
- **SR-IOV does not care about nested virtualization**



SR-IOV Emulation for Container in VM

Solution: **emulate SR-IOV**

- Implements the control path in the software
- The data path can be still offloaded with vDPA (virtio data path acceleration)



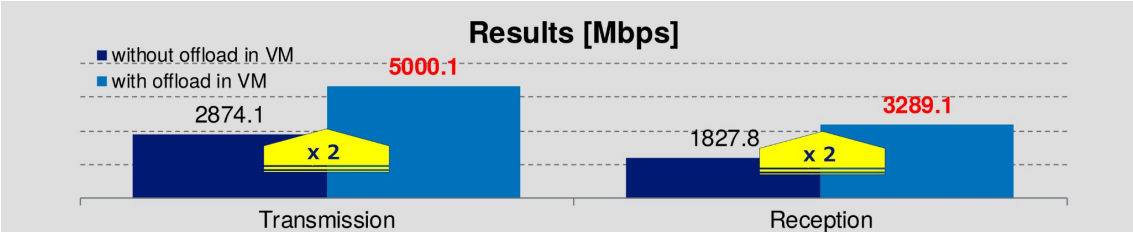
Performance Evaluation

Confirmed performance improvement
by offloading virtual network packet switching to the hardware

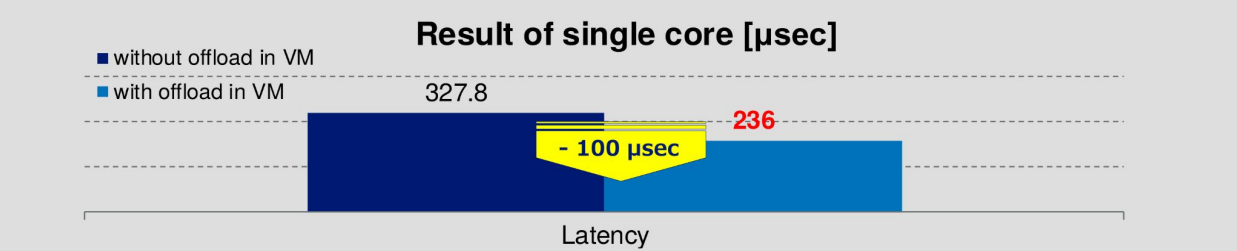
- Implemented SR-IOV emulation on QEMU
- The host configures packet switching
  - The current implementation lacks the configuration mechanism for guest
- Measured throughput and latency between:
  - External host
  - Container on VM

Performance Evaluation

Doubled UDP bulk transfer throughput



⅔ UDP round trip latency



History of SR-IOV emulation in VIRTIO/QEMU

2014: igb patch series for QEMU by Knut Omang

igb: a network device (Intel 82576)

2019: virtio version 1.1 specification with SR-IOV support

2022: nvme upstreamed by Lukasz Maniak

2023: igb upstreamed by Akihiko Odaki (details on daynix.github.io)

2023: virtio-net-pci RFC by Yui Washizu

2024: virtio-net-pci for upstreaming by Akihiko Odaki (in progress)

Adding VF controllability to VIRTIO

Simply emulating SR-IOV does not automatically add **VF controllability**

VIRTIO 1.3 draft adds a concept of **device groups**

- *Group administration commands* control devices in a device group
- *Administration virtqueue* transports group administration commands

[RFC] virtio-net: support access and control the member devices

- Allows inspecting the status
- Allows setting MAC addresses

L2/L3 packet switch offloading (switchdev)

Conclusion

Software: **receive steering**

- **Available now**: tuntap automq, QEMU/eBPF RSS
- **In progress**: hash reporting
- **Future**: inner header hash, flow filter, RSS context

Hardware: **SR-IOV**

- The **container-in-VM** scenario results in nested network virtualization
- Employ **SR-IOV emulation** to allow configuring offloading in VM
- **VIRTIO 1.3 device groups** provides a foundation for SR-IOV features

Acknowledgement

Yui Washizu (NTT Open Source Software) provided figures of the “container in VM” scenario and benchmark results. The figures and benchmark results will also be presented in KVM Forum 2024.

Appendix: Verification target's setup

Confirmed 2x performance improvement with vDPA in the following setup:

Server model	HPE ProLiant DL360 Gen10
CPU	Intel Xeon CPU 4210R @2.4GHz
NIC	Mellanox Technologies MT27710 family ConnectX-6 Dx (100G)
Host/Guest OS	Rocky Linux 9.2
QEMU version	QEMU 8.1.1 (w/ virtio SR-IOV emulation patch applied)
Kubernetes version	1.27.6
CNI plugin	Calico v3.26.3
SR-IOV CNI plugin	2.7.0 (*)
netperf	2.7

(*) with slight modification to adapt to virtio's sysfs

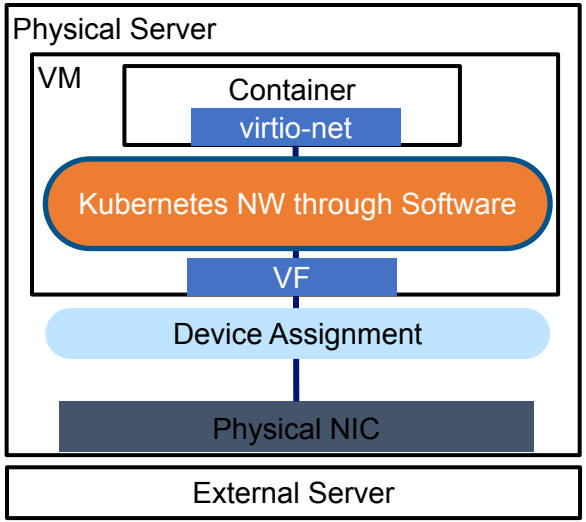
Environments

Compare the following 2 environments

- Using SR-IOV VFs as the backends and netperf as a measurement tool

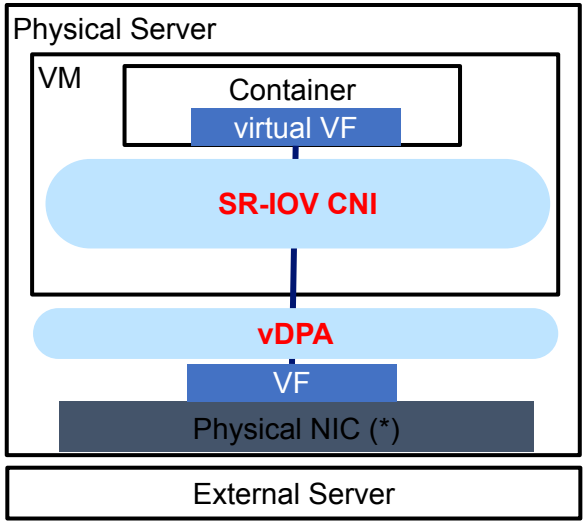
Without offloading in VM

- Baseline



With offloading in VM

- SR-IOV CNI configures virtual VFs

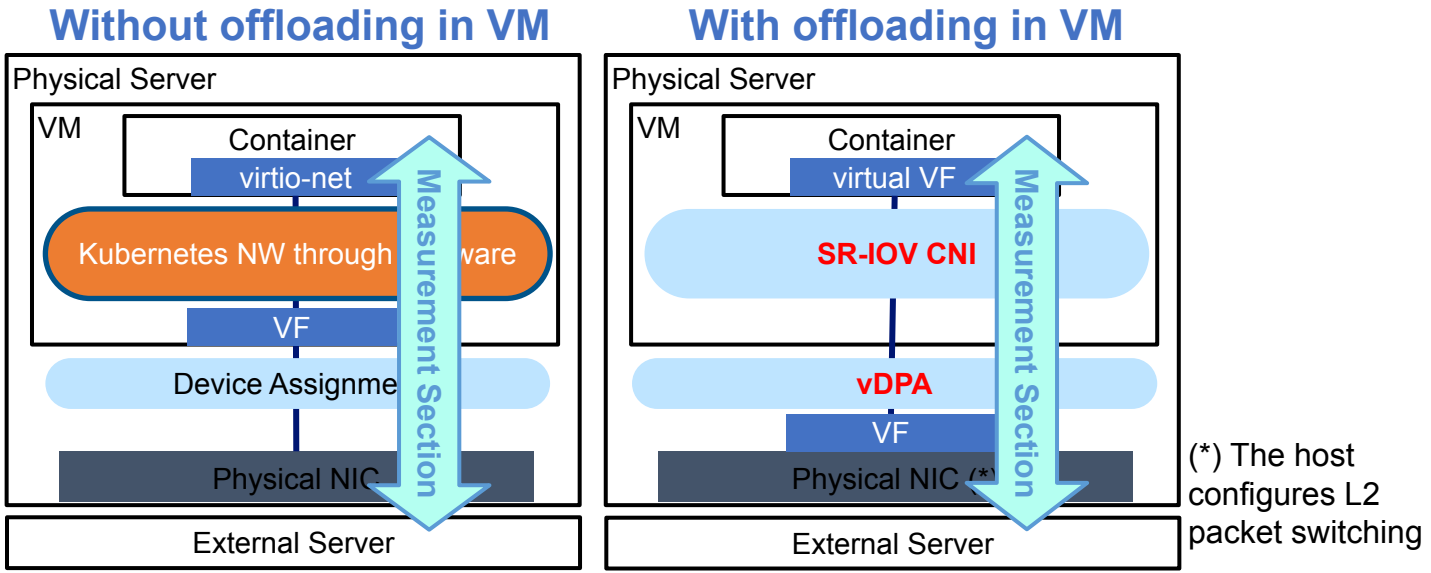


(*) The host configures L2 packet switching

Metrics and section

Verification metrics are throughput and latency

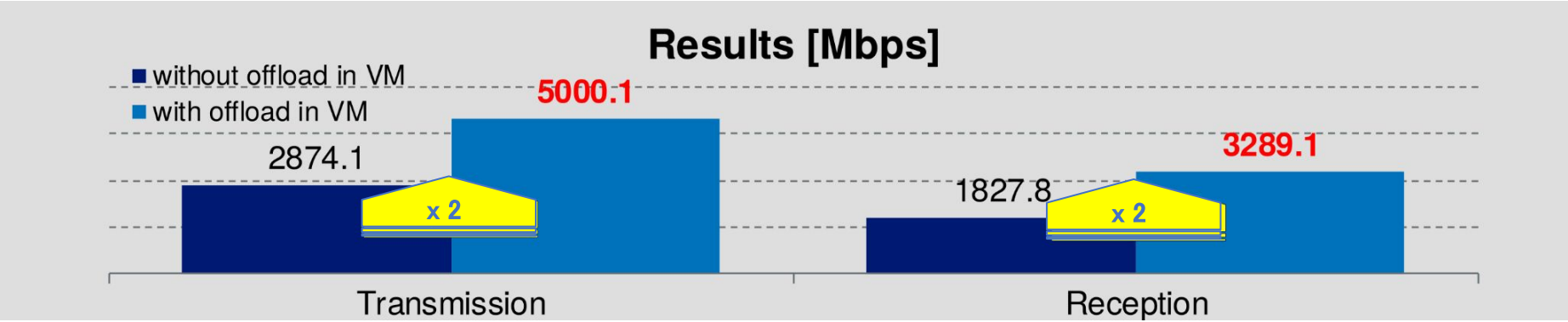
- Metrics: Throughput and latency
- Section: Between a container on the VM and an external machine



(*) The host configures L2 packet switching

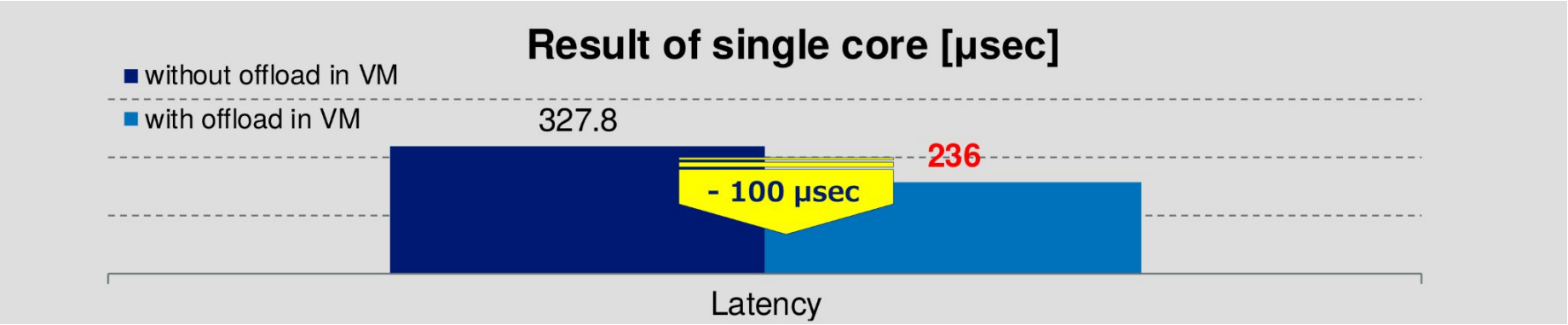
Throughput

- Measuring method
  - Average throughput of UDP bulk transfer with netperf
- Results
  - Transmission: **x 2** (2874.1 Mbps → 5000.1 Mbps)
  - Reception: **x 2** (1827.8 Mbps → 3289.1 Mbps)



Latency

- Measuring method
  - 99%ile of UDP round trip time using netperf
- Results
  - **- 100 µsec** (327 µsec → 236 µsec)



Virtualization

Two main components:

- Linux/KVM for low-level virtualization operation
  - Uses privileged/processor-specific features
- VMM for high-level operations
  - Implements virtio as a paravirtualized interface
  - Makes VM look like a “machine”
  - I assume QEMU in this presentation