KVM Forum 2024, Brno, Czech:

Unleashing SR-IOV on Virtual Machines

Yui Washizu

NTT Open Source Software Center
yui.washidu@gmail.com

Akihiko Odaki

Daynix Computing Ltd.
akihiko.odaki@daynix.com

Introduction

Unleashing SR-IOV on Virtual Machines

Multi-tenant cloud environments

Two goals of multi-tenant cloud environments:

- **Cost-effective**
  - A primary motivator for multi-tenant cloud deployments
  - Maximize resource utilization by eliminating the need for infrastructure dedicated to each tenant
- **Secure**
  - Each tenant is independent
  - Each tenant can only view and configure its own resources

Single Root I/O Virtualization (SR-IOV)

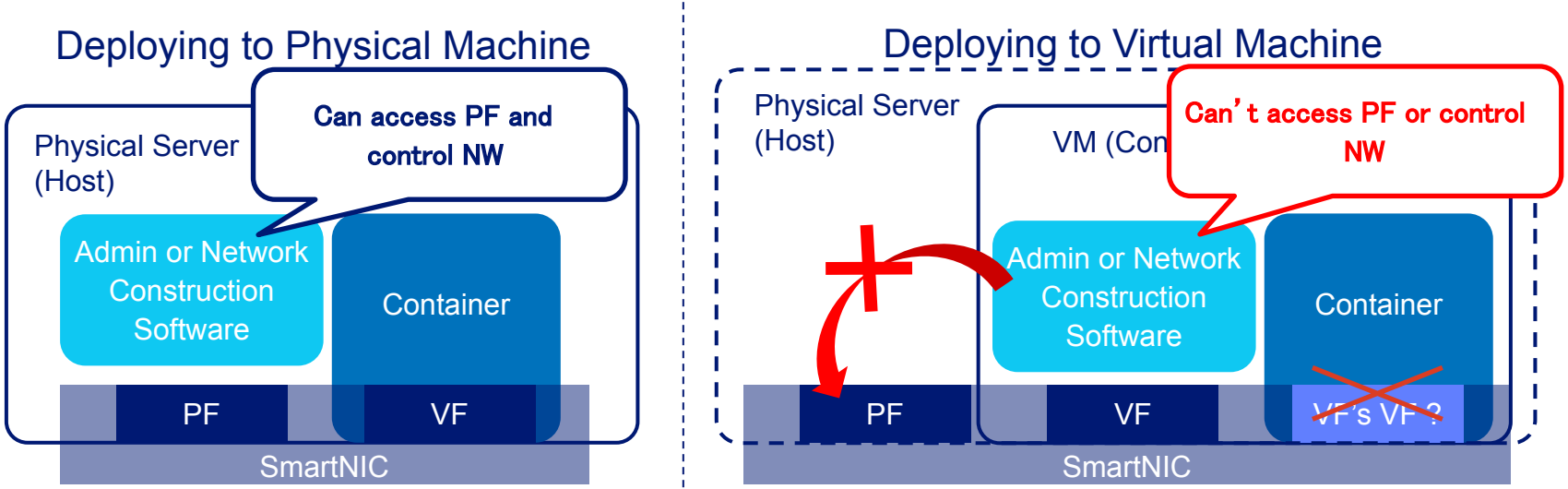
Achieve the two goals of multi-tenant cloud environment

- A single PCIe device presents Physical and Virtual Functions
  - **P**hysical **F**unction (PF) allows the host configuring VFs
  - **V**irtual **F**unction (VF) is independently assigned to containers or VMs
- Cost-effective: Reduce network virtualization overhead
  - Configure the offloading of packet switching with the PF
- Secure: Network functions are allocated exclusively
- A problem arises in some situations (next slide)

Problem with offloading container networks on VMs

Containers on VMs require their own virtual network

- However, VMs cannot offload such a virtual network to hardware
  - › The host controls the PF exclusively to construct a virtual network for VMs



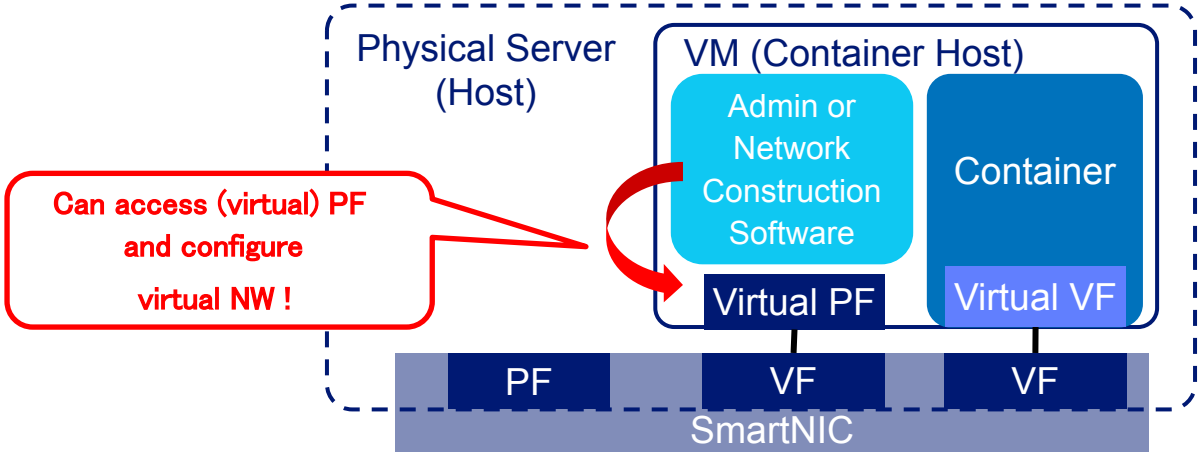
Our proposal

Unleashing SR-IOV on Virtual Machines

Proposal: SR-IOV emulation

Emulate SR-IOV with VMM

- VMs gain their own PFs (="**virtual PFs**") and corresponding VFs (="**virtual VFs**")
- VMs can configure network offloading through virtual PFs



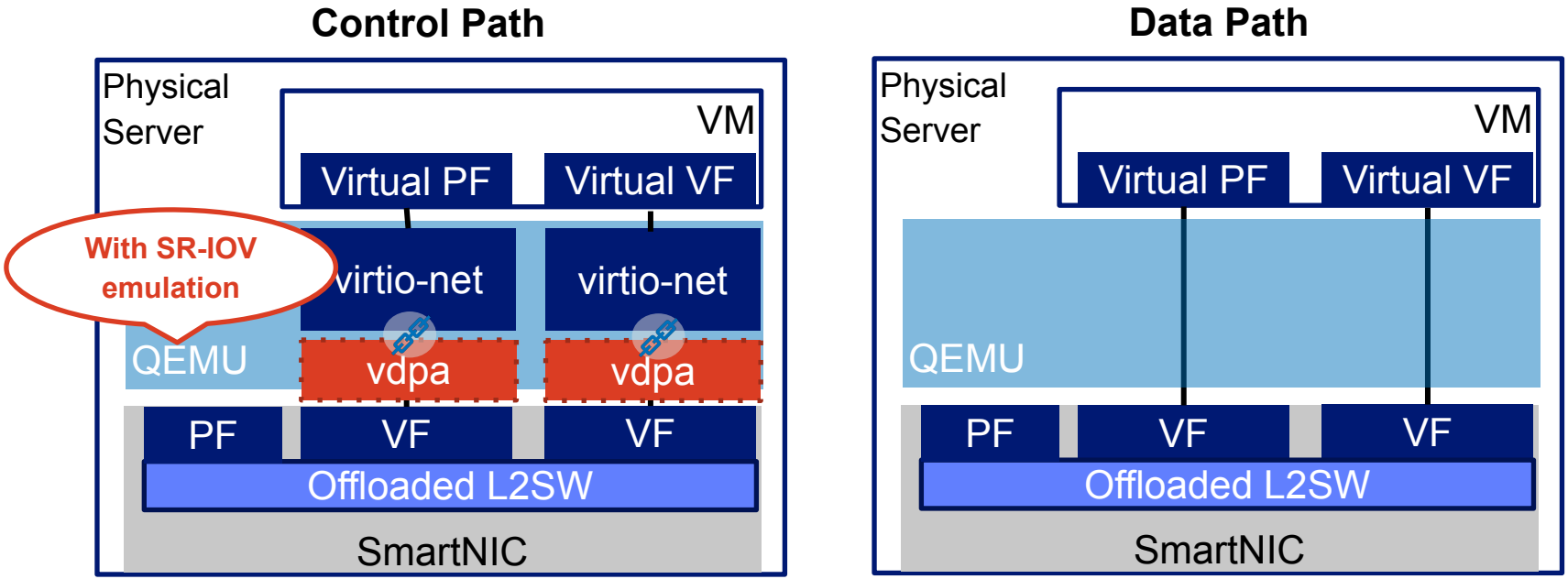
Advantages of SR-IOV emulation

- Consistent with bare-metal system
  - Allows the use of existing network construction software (e.g. SR-IOV CNI plugin)
- Scalable: Eliminates the need to assign one PF per VM
- Secure: Users cannot control entire NIC hardware from within a VM

Avoiding emulation overhead with vDPA

SR-IOV emulation solely governs the control path

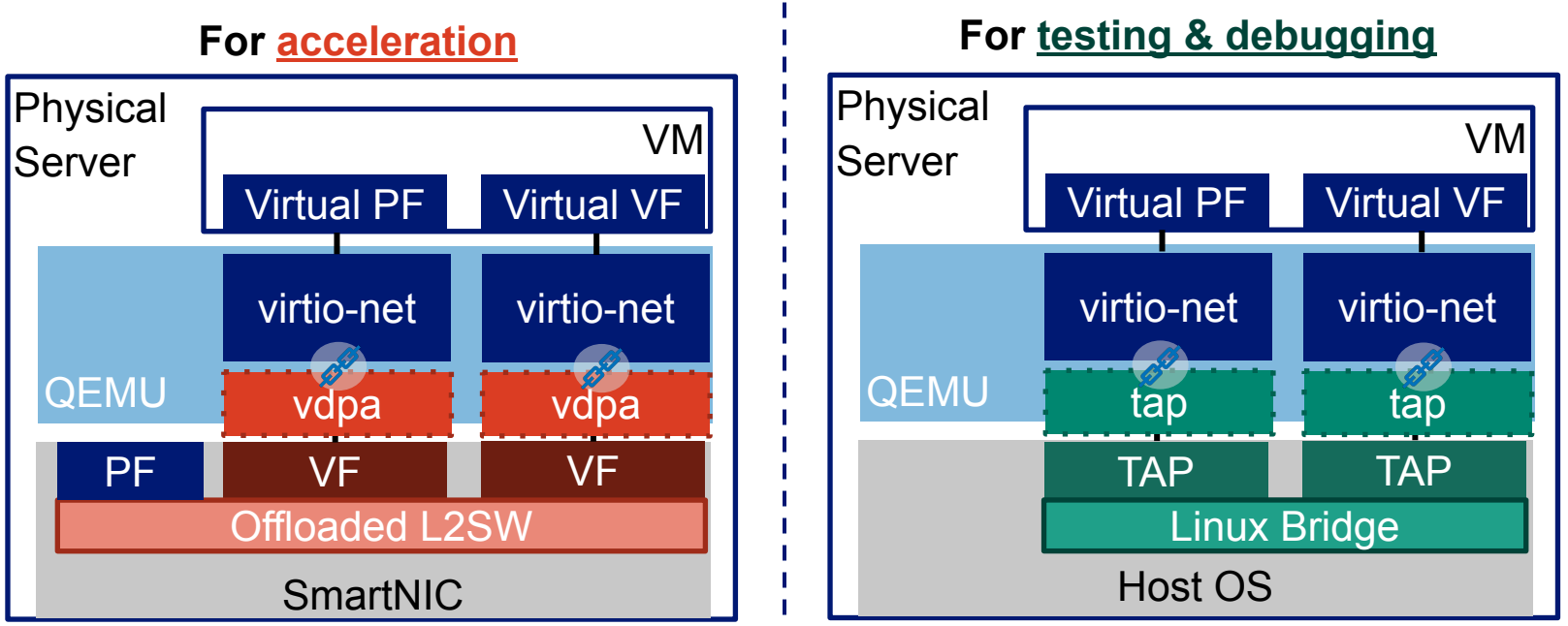
- The data path can be still offloaded with vDPA (virtio Data Path Acceleration)



Adapting SR-IOV emulation to other use cases

Replace backends for other use cases

- E.g., use tap/user devices to test virtio-net driver with SR-IOV for the guest OS



Interface for virtio SR-IOV embedded switch

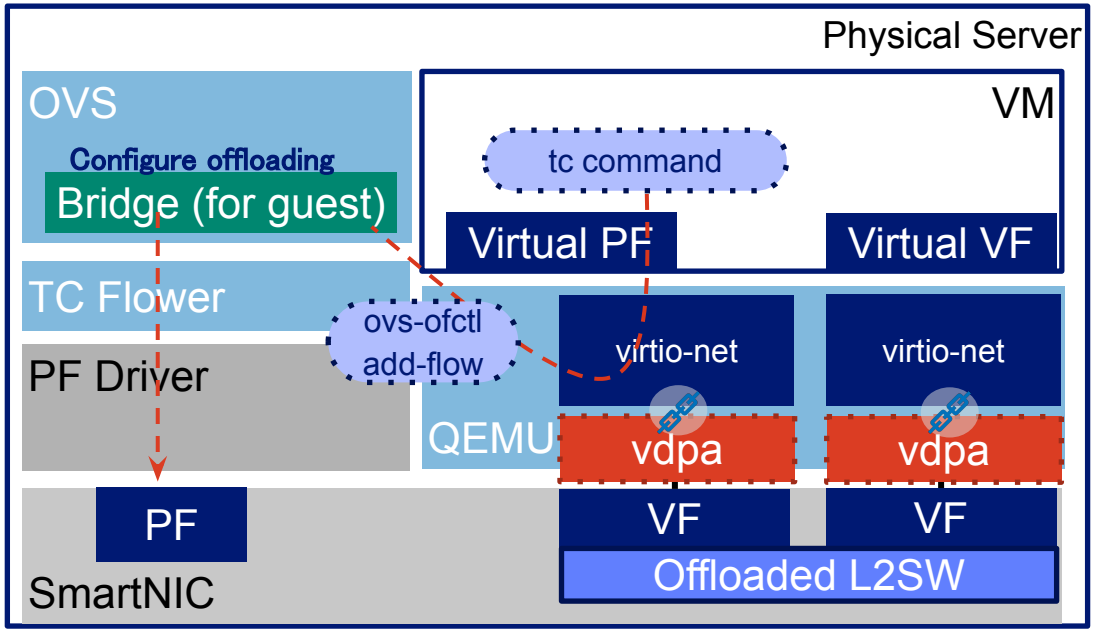
virtio 1.3 will expose SR-IOV embedded switch capability as device groups

- A device group allow its owner device to control its member devices
- Some features using device groups are already proposed:
  - E.g., [virtio-comment] [RFC] virtio-net: support access and control the member devices
    - Inspect the statuses of VFs
    - Set MAC addresses of VFs
- Extend virtio spec for packet switching based on device groups

Future work: offload packet switching

Provide comprehensive solution of NW offloading on VMs with OVS

- QEMU processes TC configuration from within VMs
- Host's OVS configures offloading
  - A bridge in OVS accommodates virtual PF and VFs



Performance Verification

Unleashing SR-IOV on Virtual Machines

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

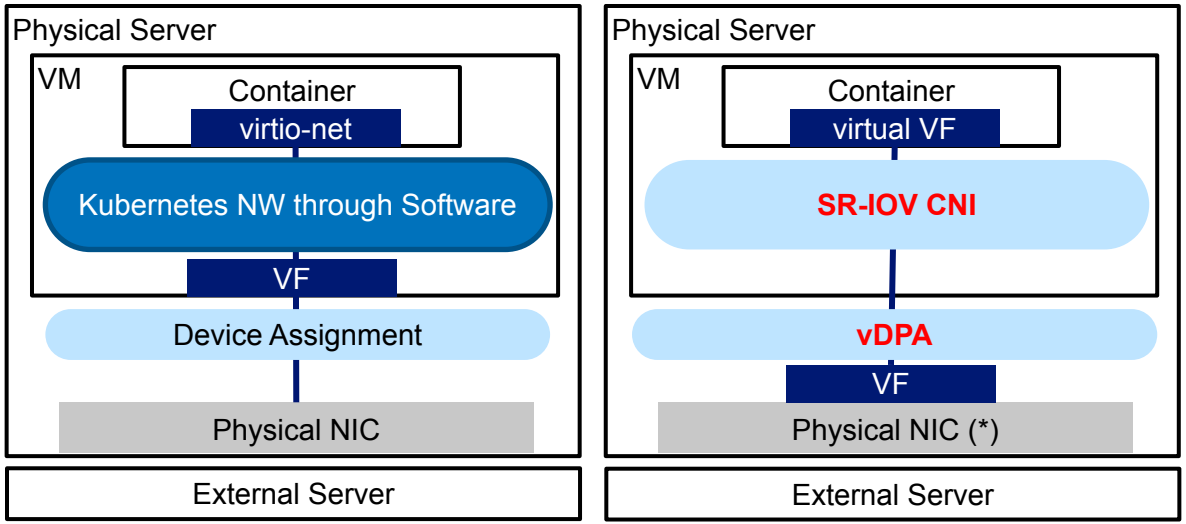
- Using SR-IOV VFs as the backends and netperf as a measurement tool

Without offloading in VM

- Baseline

With offloading in VM

- SR-IOV CNI configures virtual VFs

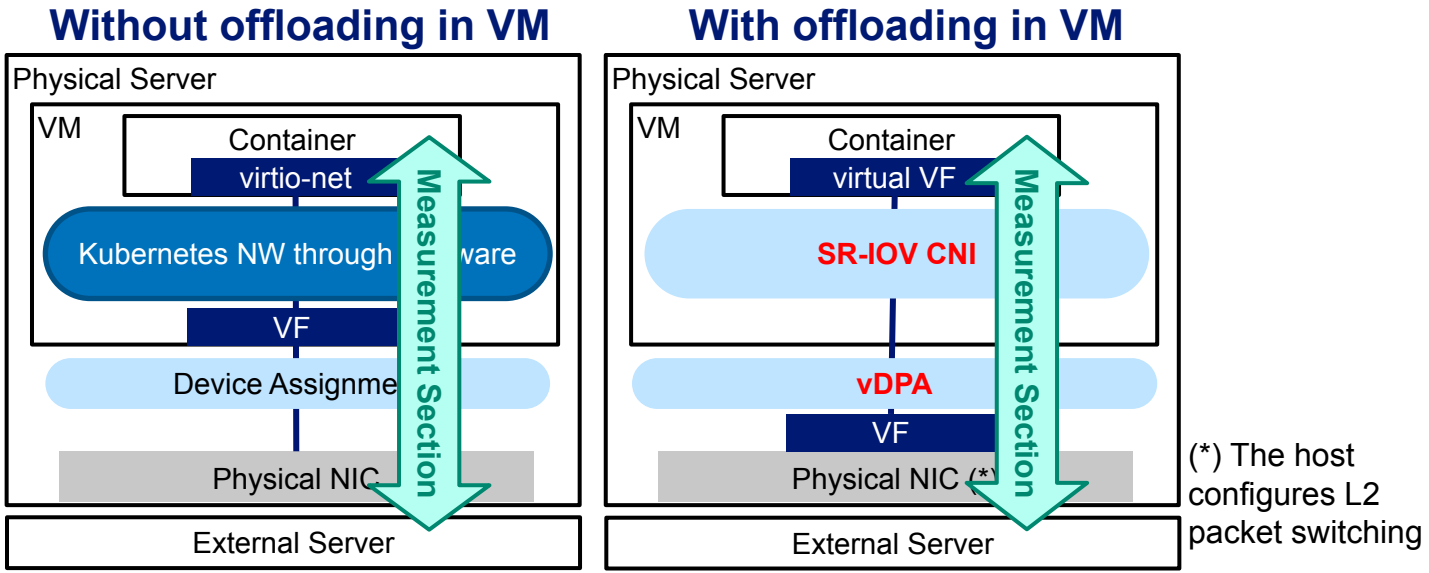


(*) The host configures L2 packet switching

Metrics and section

Verification metrics are throughput and latency

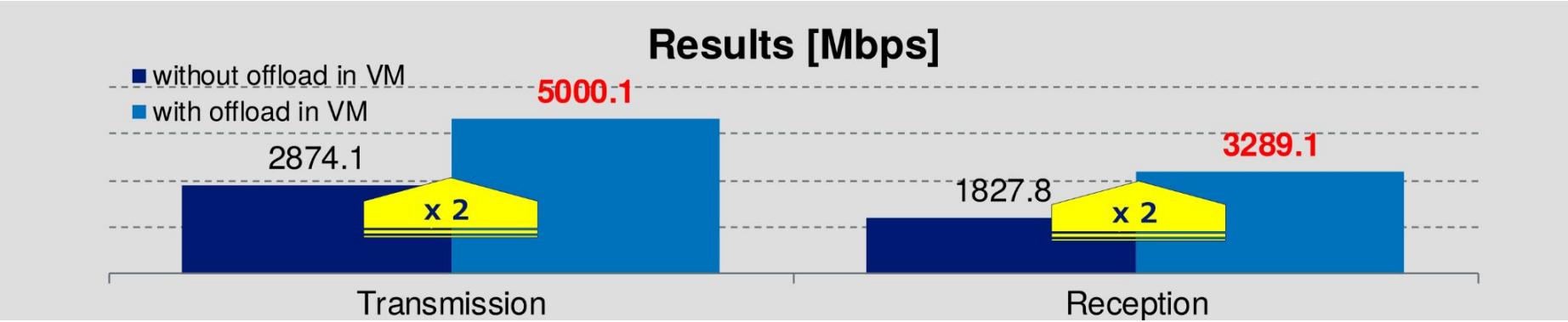
- Metrics: Throughput and latency
- Section: Between a container on the VM and an external machine



(*) The host configures L2 packet switching

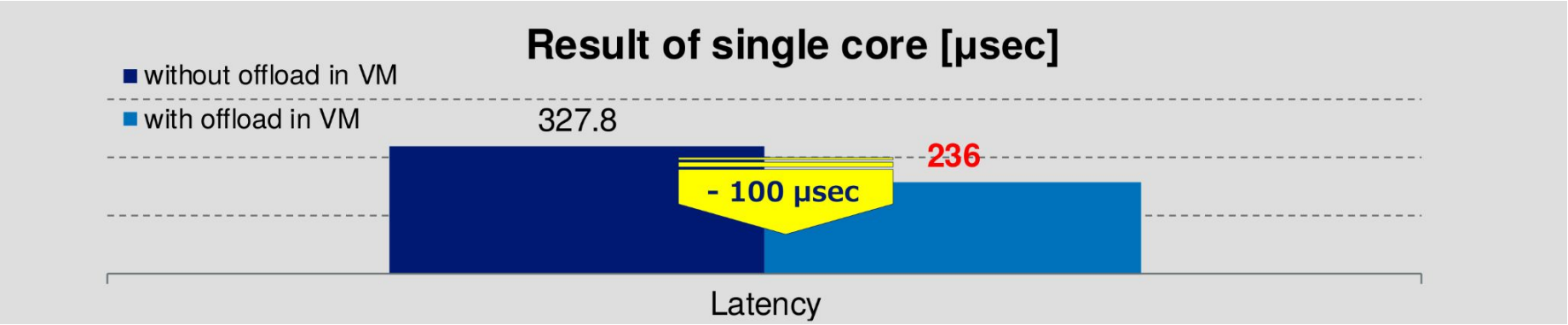
Throughput

- 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)



Development of SR-IOV Emulation in QEMU

Unleashing SR-IOV on Virtual Machines

History of SR-IOV emulation

2014: igb patch series 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 SR-IOV to virtio-net-pci

Paravirtualized

- igb: physical device emulation

Challenge: flexible configuration

- Varying number of VFs
  - igb: Fixed number of VFs
- Network backends (`netdev`)
  - igb: One backend and hardware-defined packet switching

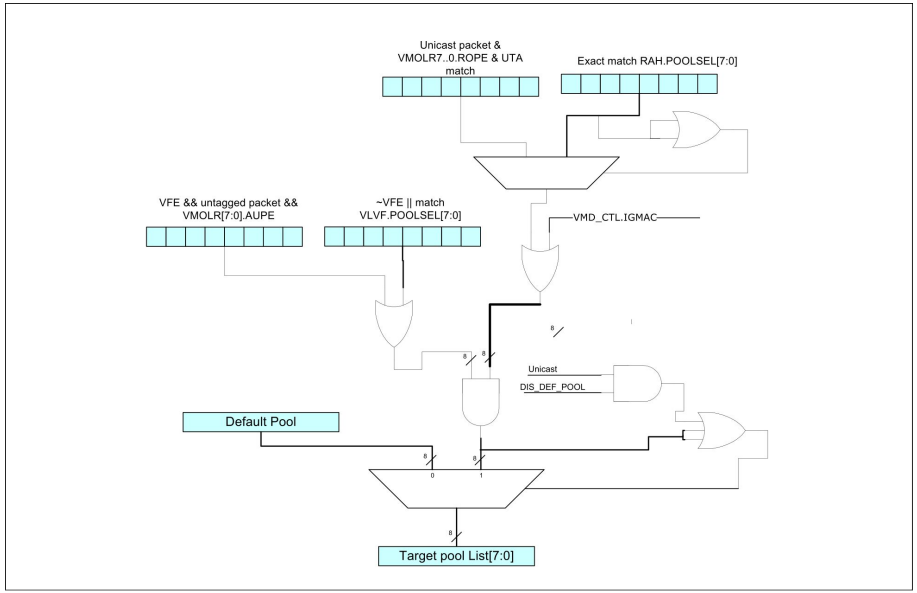
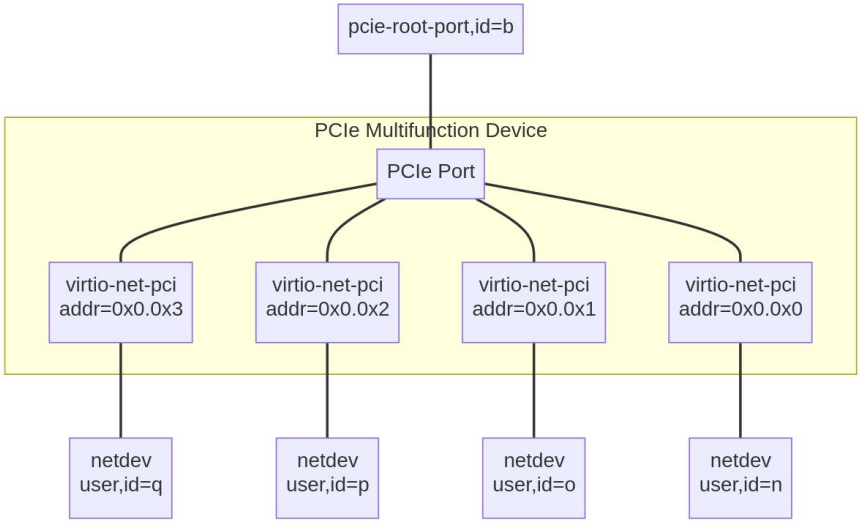


Figure 7-29. Pool List Selection — Replication Disabled

Conventional PCI multifunction

Just specify **`multifunction`** and **`addr`**:

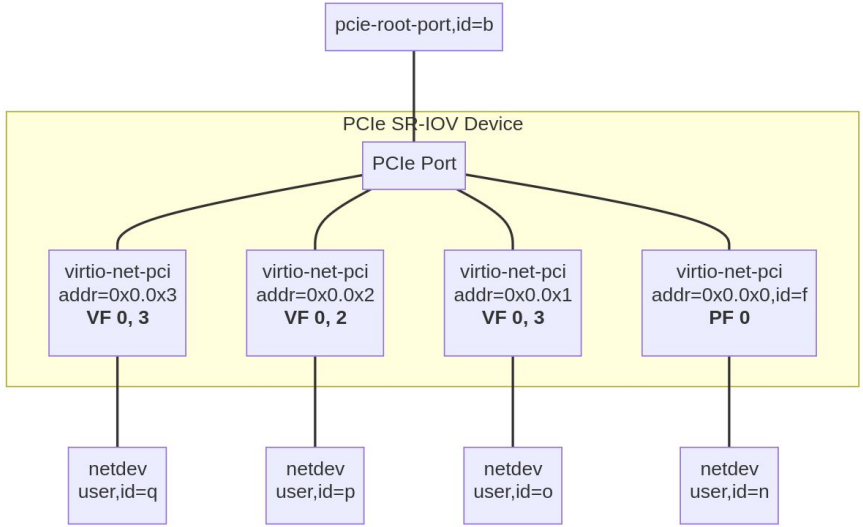
```
-netdev user,id=n -netdev user,id=o
-netdev user,id=p -netdev user,id=q
-device pcie-root-port,id=b
-device virtio-net-pci,netdev=q,bus=b,
        addr=0x0.0x3
-device virtio-net-pci,netdev=p,bus=b,
        addr=0x0.0x2
-device virtio-net-pci,netdev=o,bus=b,
        addr=0x0.0x1
-device virtio-net-pci,netdev=n,bus=b,
        addr=0x0.0x0,multifunction=on
```



Composable SR-IOV device

Add the **`sriov-pf`** property:

```
-netdev user,id=n -netdev user,id=o
-netdev user,id=p -netdev user,id=q
-device pcie-root-port,id=b
-device virtio-net-pci,netdev=q,bus=b,
        addr=0x0.0x3,sriov-pf=f
-device virtio-net-pci,netdev=p,bus=b,
        addr=0x0.0x2,sriov-pf=f
-device virtio-net-pci,netdev=o,bus=b,
        addr=0x0.0x1,sriov-pf=f
-device virtio-net-pci,netdev=n,bus=b,
        addr=0x0.0x0,id=f
```



The implementation is a bit more complicated though.

SR-IOV as guest-controlled hotplugging

The VF lifetime is controlled by the guest

1. The guest configures the resource allocation in the device
2. The guest enables VFs

Similar to hotplug, but the guest expects:

- Function numbers are already allocated
- Underlying resources (e.g., `netdev`) are already allocated

…because physical devices do

Issues with hotplugging

Today: literally hotplugging VFs as the guest requests.

- **Problem 1**: function numbers and resources are not reserved
- **Problem 2**: ad-hoc CLI
  - The conventional CLI immediately plugs VFs instead of hotplugging

```
-netdev user,id=n -netdev user,id=o
-netdev user,id=p -netdev user,id=q
-device pcie-root-port,id=b
-device virtio-net-pci,bus=b,addr=0x0.0x3,netdev=q,sriov-pf=f
-device virtio-net-pci,bus=b,addr=0x0.0x2,netdev=p,sriov-pf=f
-device virtio-net-pci,bus=b,addr=0x0.0x1,netdev=o,sriov-pf=f
-device virtio-net-pci,bus=b,addr=0x0.0x0,netdev=n,id=f
```

Avoiding hotplugging

[PATCH v16 00/13] hw/pci: SR-IOV related fixes and improvements:

1. **Realize the VFs when the paired PF gets realized**
   - Reserves function numbers and resources
   - VFs can be added with normal `-device` command line
2. **But leave them *disabled***
   - Reuses the code to power down PCI devices
3. **Enable them when the guest requests**

Validation of SR-IOV device configuration

SR-IOV imposes several restrictions:

- SR-IOV requires PCI Express.
- A function cannot be a PF and VF at the same time.
- A pair of PF and VFs must be on the same bus.
- The vendor and device ID of the VFs must be consistent.
- The memory region configurations of the VFs must be consistent.
- The IDs of VFs must be linear (i.e., have a consistent stride).
- VFs do not implement Expansion ROM.

To satisfy these requirements:

- Check them when composing a SR-IOV device[1][2][3]
- Limit SR-IOV device composition to virtio-net-pci as a precaution[4]

Summary

Unleashing SR-IOV on Virtual Machines

Summary

- Background: virtual network offloading with SR-IOV
  - Emulating SR-IOV can provide configurability to the guest for containers
  - Proposals to exploit the emulated SR-IOV capability
    - [virtio-comment] [RFC] virtio-net: support access and control the member devices
    - Even packet switching?
  - The SR-IOV emulation will be immediately useful for testing
- Benchmarks show a big win with offloaded packet switching:
  - Throughput (Tx): **x 2** (2874.1 Mbps → 5000.1 Mbps)
  - Throughput (Rx): **x 2** (1827.8 Mbps → 3289.1 Mbps)
  - Latency: **- 100 µsec** (327 µsec → 236 µsec)
- We aim to land the SR-IOV emulation for virtio-net-pci in QEMU 9.2

Introduction

Unleashing SR-IOV on Virtual Machines

Multi-tenant cloud environments

Two goals of multi-tenant cloud environments:

- **Cost-effectiveness**
  - A primary motivator for multi-tenant cloud deployments
  - Maximize resource utilization and avoid the need for dedicated infrastructure to each tenant
- **Security**
  - Each tenant is independent
  - Each tenant can only view and configure their own resources

Single Root I/O Virtualization (SR-IOV)

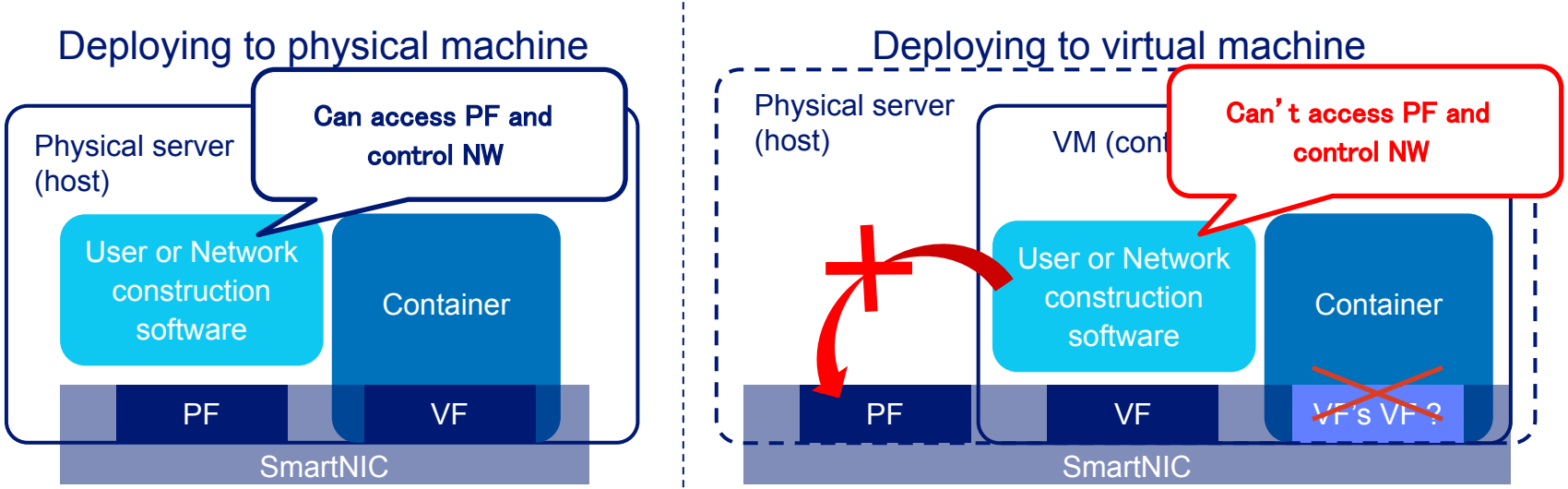
A single PCIe device presents Physical and Virtual Functions

- Roles of the Physical Function and Virtual Functions:
  - **P**hysical **F**unction (PF) allows the host configuring VFs through PFs
  - **V**irtual **F**unctions (VFs) are assigned to containers or VMs
- Reduce network virtualization overhead
  - Configure the offloading of packet switching with the PF
- A problem arises in a specific workload

The problem with offloading container networks on VMs

VMs cannot control networks by accessing the SR-IOV PF

- Objective: Virtual network for containers
- Container virtual networks cannot be offloaded using SR-IOV
  - PF is only available on physical machine



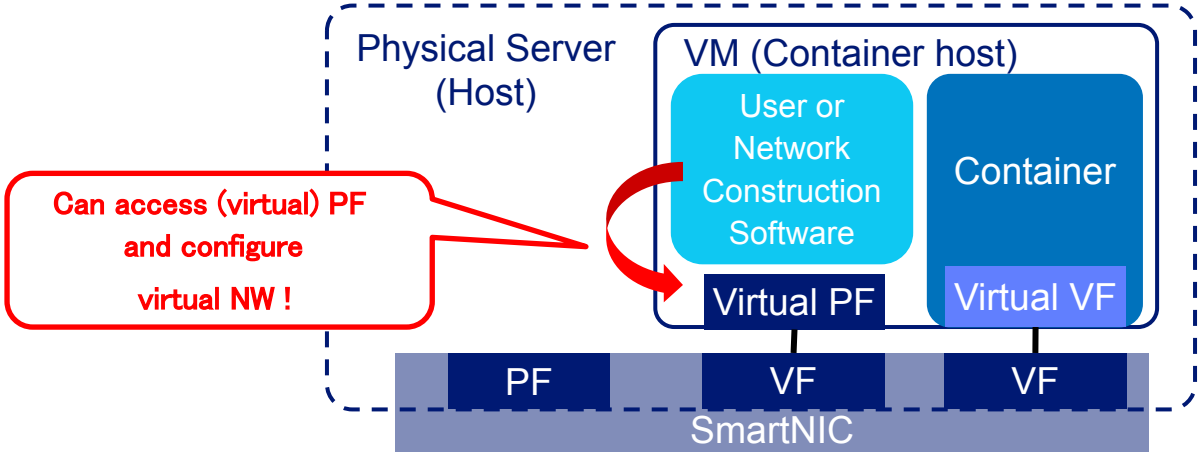
Our proposal

Unleashing SR-IOV on Virtual Machines

Proposal: SR-IOV emulation

Emulate PCIe devices with SR-IOV on QEMU

- Allow using virtio SR-IOV in VMs
- Able to create new VFs (=**“virtual VFs”**) through a PF (=**“virtual PF”**) in a VM
- Also able to handle hardware control requests from a VM through the virtual PF in the future



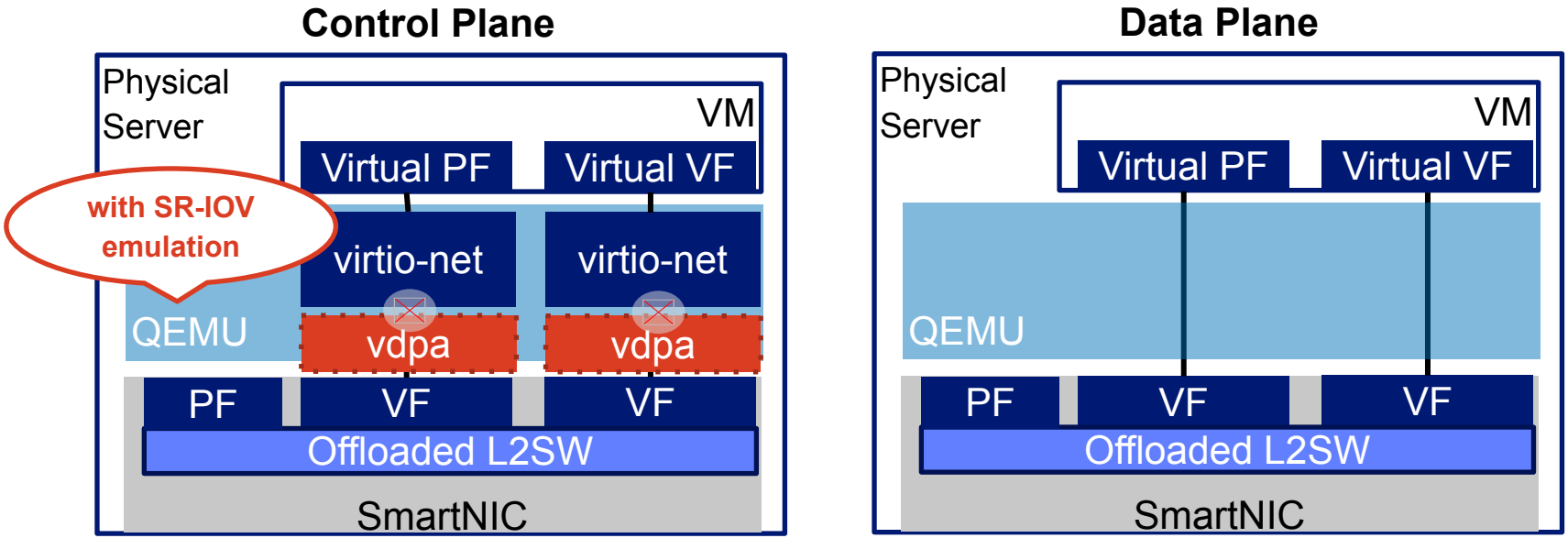
Advantage of SR-IOV emulation

- Consistent with bare-metal system
  - Allows the use of existing network construction software (e.g. SR-IOV CNI plug-in)
  - Facilitates the offloading of more advanced network features in the future
- Scalable: Eliminates the need to assign one PF per VM
- Secure: Users cannot control entire NICs hardware from within a VM

Offload container networks with SR-IOV emulation

Combine with Virtio Data Path Acceleration (vDPA)

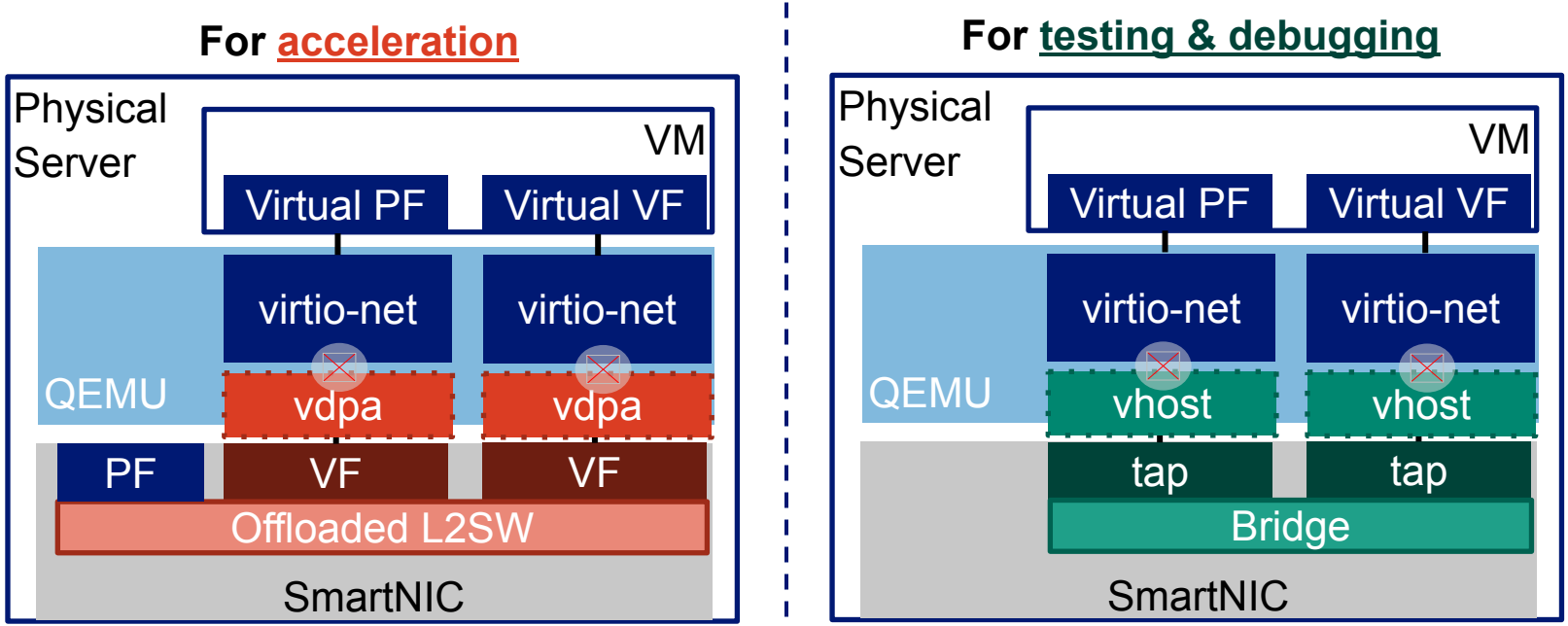
- vDPA offloads only the data plane
- Allows the host‘s VF to exist as a virtual PF within a VM



Adapting SR-IOV emulation to other use cases

Replace backends for other use cases

- E.g., use tap devices to test guest OS with SR-IOV



Implement device groups for virtio with SR-IOV

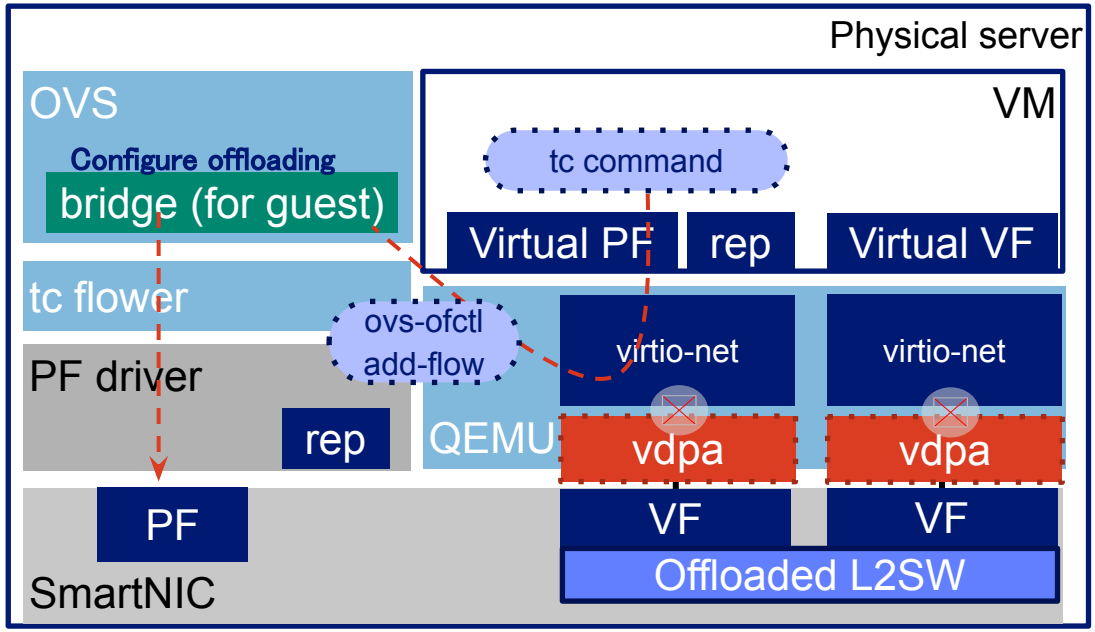
Help to implement the features related to virtio SR-IOV

- Features using device groups
  - Device groups allow a device to control a group of other devices
  - Currently, only SR-IOV included as a device group type
    - SR-IOV needs to be used in VMs
  - The following suggestions exist in the virtio community:
    - [virtio-comment] [RFC] virtio-net: support access and control the member devices
    - [virtio-comment] [PATCH v15] admin: Add group member legacy register access commands
    - [virtio-comment] [PATCH v4 0/1] VIRTIO: Introduce MGMT device and Provision maximum MSI-X vectors for a VF

Future work: offload advanced NW features

Potential solution for advanced network features on VMs with OVS

- QEMU (+ libvirt etc.) handles TC configuration from within VMs
- Host's OVS configures offloading
  - The bridge handles virtual PF and VFs



Performance Verification

Unleashing SR-IOV on Virtual Machines

Verification target's setup

We confirmed performance improvement with vDPA in the following setup:

Server model	HPE ProLiant DL360 Gen9
CPU	Intel Xeon CPU E5-2600 @2.3GHz
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 emulatiopatch 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

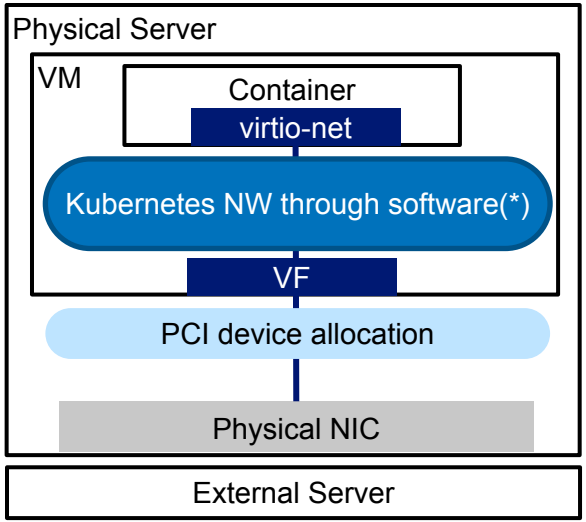
Environments

Performance verification in the following 2 environments

- Using SR-IOV VFs as the backend and netperf as measurement tool

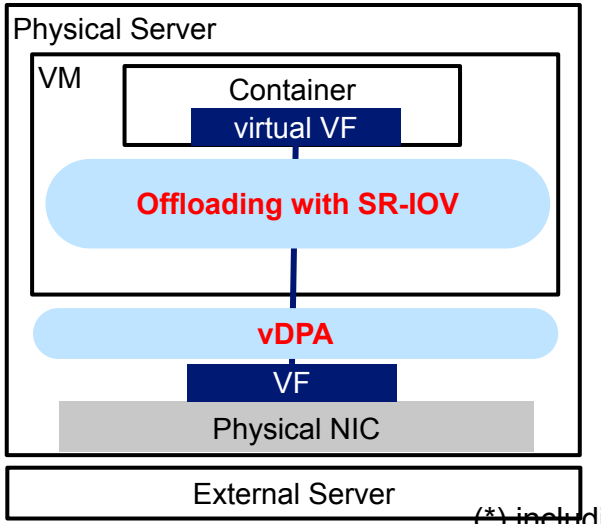
Without offloading in VM

- Baseline



With offloading in VM

- SR-IOV CNI configures HW offload on VM

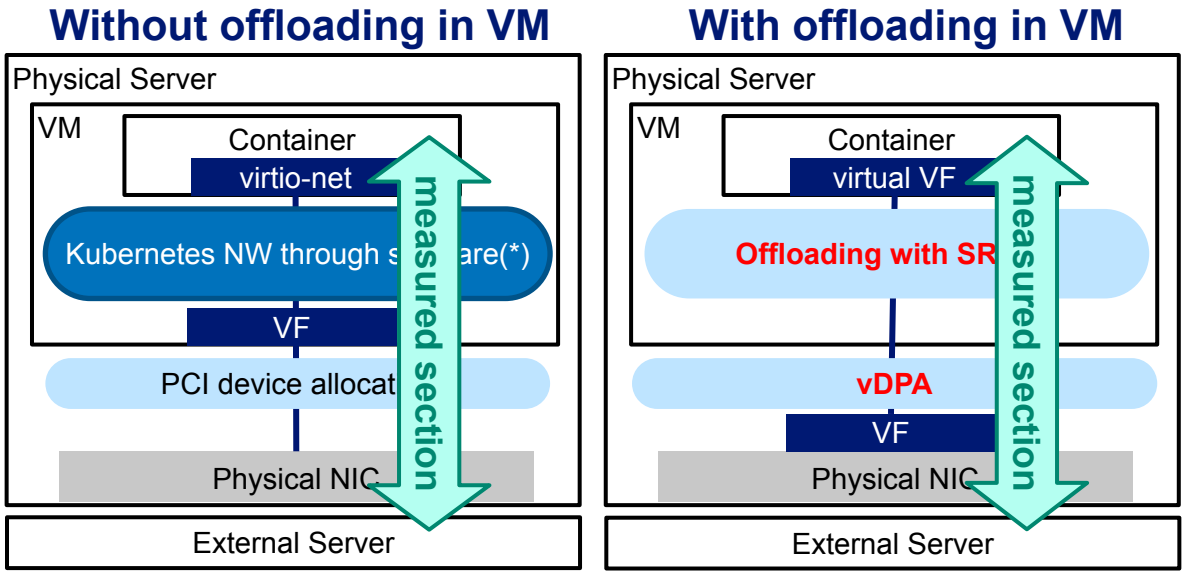


(*) including firewall and NAT

Metrics and section

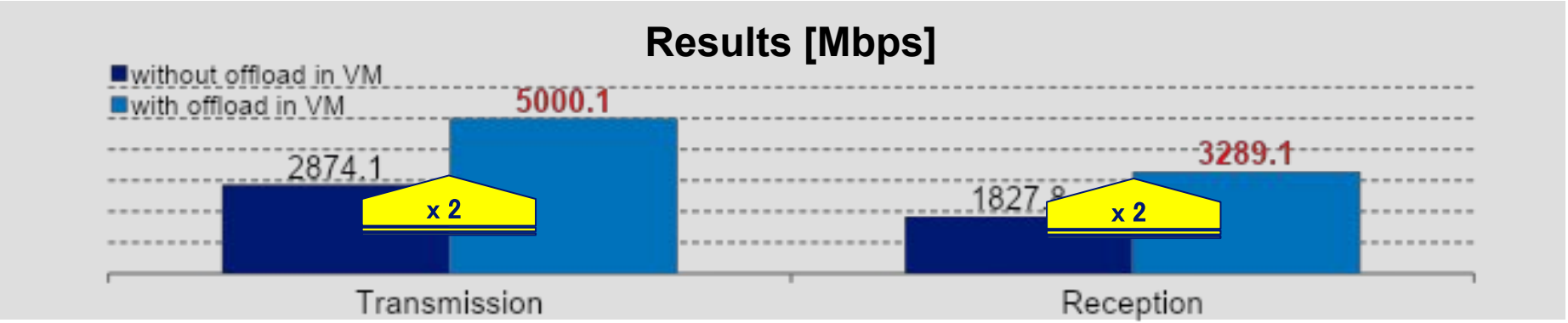
Verification metrics are throughput and latency

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



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)