Netdev 0x17, Vancouver, Canada:

Unleashing SR-IOV Offload on Virtual Machines

Yui Washizu

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

Agenda

1. Introduction
2. Approach
3. Functional/Performance Verification
4. Summary

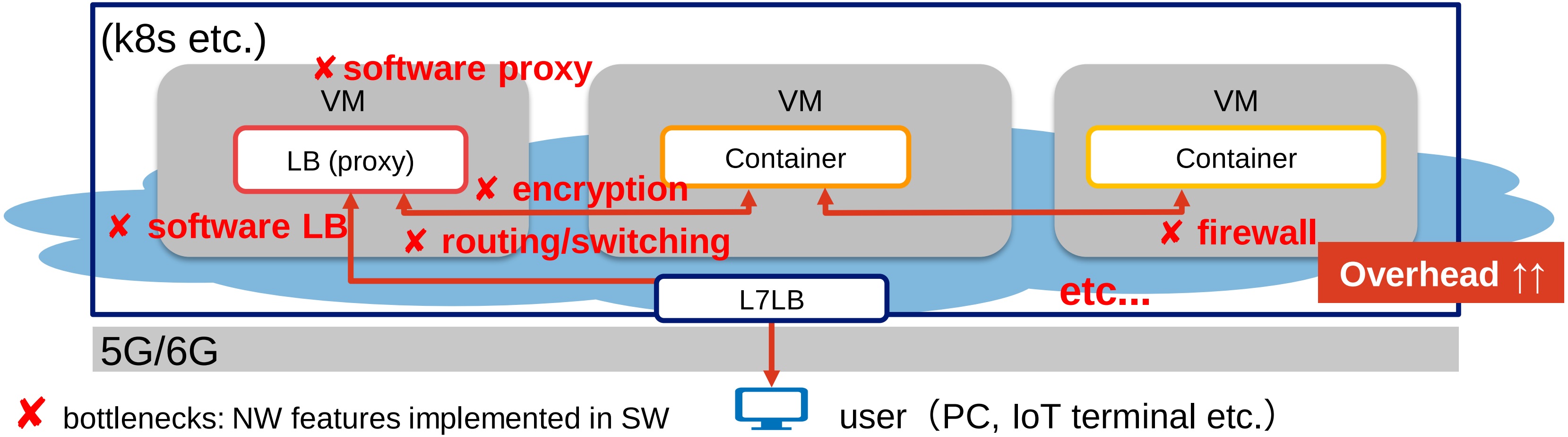
Introduction

Unleashing SR-IOV Offload on Virtual Machines

Overhead of Virtual Network

NW functions in virtual NW of VMs/containers implemented in software

- Network functions: routing, switching or encryption etc.
  - On platforms like OpenStack or Kubernetes etc.
- Software implementation causes the occurrence of overheads on emulation layer.



✗ bottlenecks: NW features implemented in SW

user（PC, IoT terminal etc.）

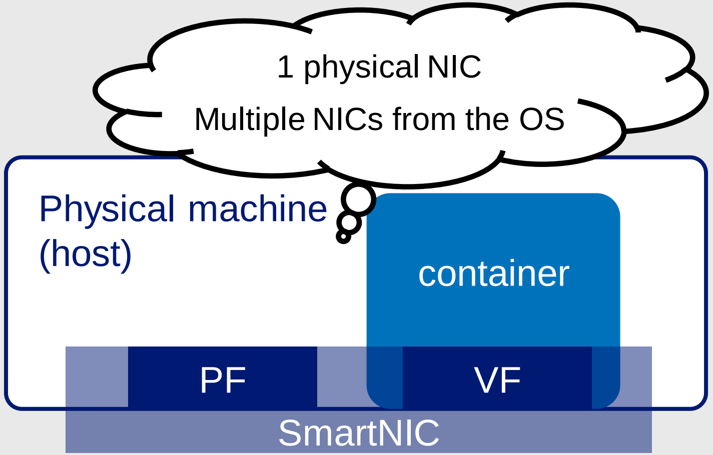
Hardware offloading

Can drastically reduce overhead of software implementation

- OpenStack and Kubernetes especially can use Single Root I/O Virtualization (SR-IOV) for hardware offloading.

What is SR-IOV ?

- enables a single PCIe function
  to present multiple separate PCI function
- called virtual functions (VFs)



Offloading on VMs

Offloading Linux network on VMs to SR-IOV physical NICs

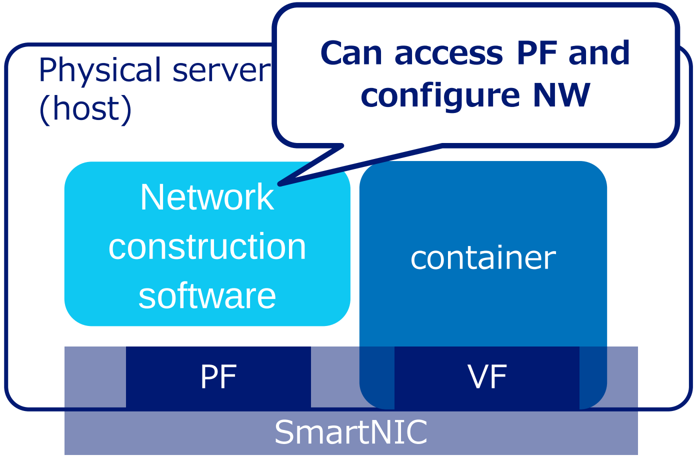
- Network offloading on VMs isn’t possible
  - Available only on physical machine
- Offloading of container network on VMs is not supported
  - Demand of deploying containers to VMs for high flexibility
- The same is true for nested VMs

Why can't networking on VMs be offloaded?

NW construction software can't access SR-IOV PF

- Network construction software configures networks by accessing SR-IOV PF
  - e.g. SR-IOV CNI plug-in (used on OpenStack or Kubernetes)

Deploying to physical machine



Can access PF and configure NW

Physical server (host)

Network construction software

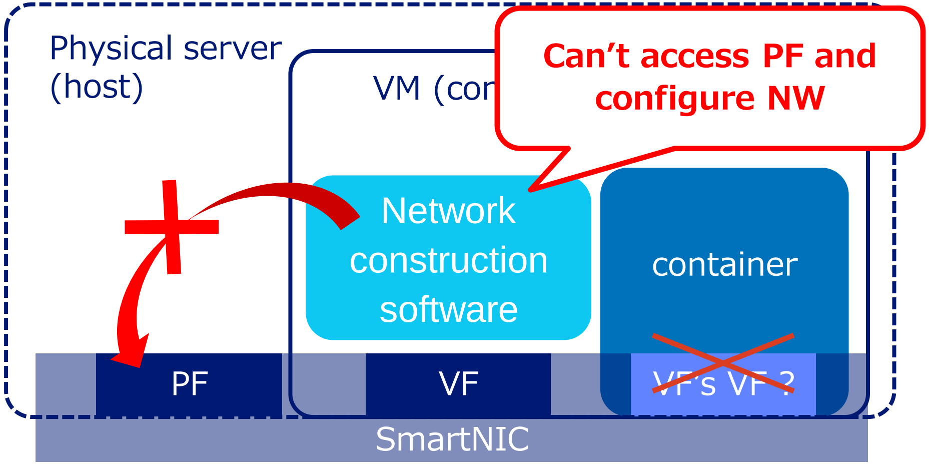
container

PF

VF

SmartNIC

Deploying to virtual machine



Can't access PF and configure NW

Physical server (host)

VM (con

Network construction software

container

PF

VF

~~VF's VF ?~~

SmartNIC

Can’t solve this by existing function ?

Following methods aren’t enough to solve this issue

- Passthrough host’s VF to VM ?
  - Can’t control SR-IOV within guest (can’t use SR-IOV CNI etc.)
  - Hard to add feature of switchdev SR-IOV on guest
    - Can’t create rep device for switchdev mode without accessing to PF within VM
- Assign PFs to VM and allow VMs to exclusively access PFs ?
  - Lack scalability and introduce security concern

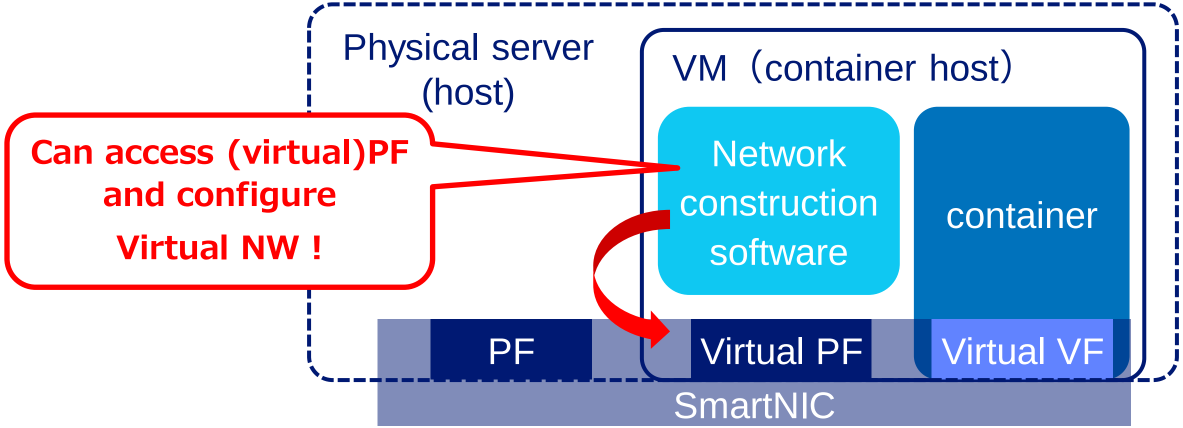
Approach

Unleashing SR-IOV Offload on Virtual Machines

Proposed Approach: Virtual PF

Emulates PFs that have SR-IOV feature (=“virtual PF”)

- **Virtual PFs** (implemented mainly in Qemu)
  - Allow host’s SR-IOV to be controlled within the guest (SR-IOV emulation)
  - Handle hardware control requests from the guest through the virtual PF
    - To configure the hardware and create new VFs (=”**virtual VF**”)



SR-IOV emulation

SR-IOV emulation’s advantage:

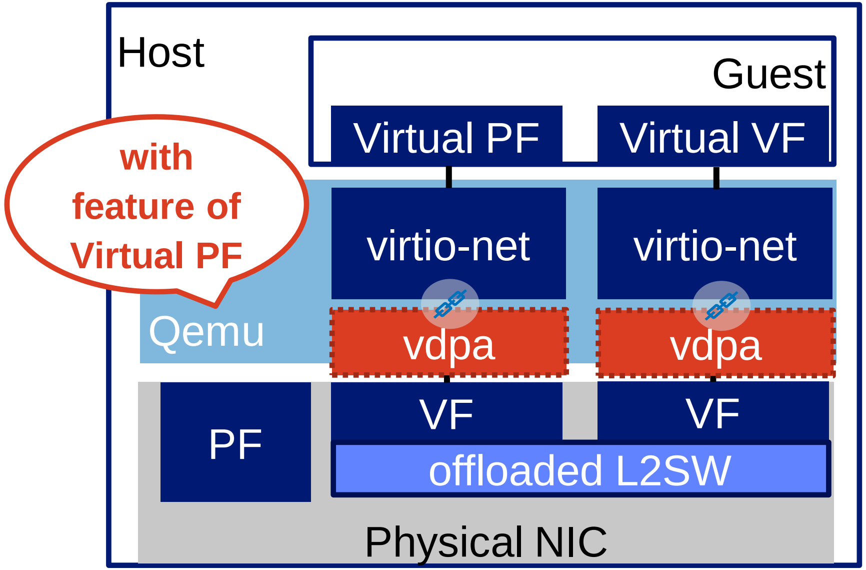
- Same way as the way on physical machine
  - Application works same as on physical machine
    - Available to use existing network construction software (e.g. SR-IOV CNI plug-in)
  - Create rep device for switchdev mode (in the future)
- Scalability and less security concern
  - Not directly access to PFs
  - Scalability: Needless to assign one PF per VM
  - Less security concern: guests can’t control entire NIC hardware

How to accelerate NW by Virtual PF

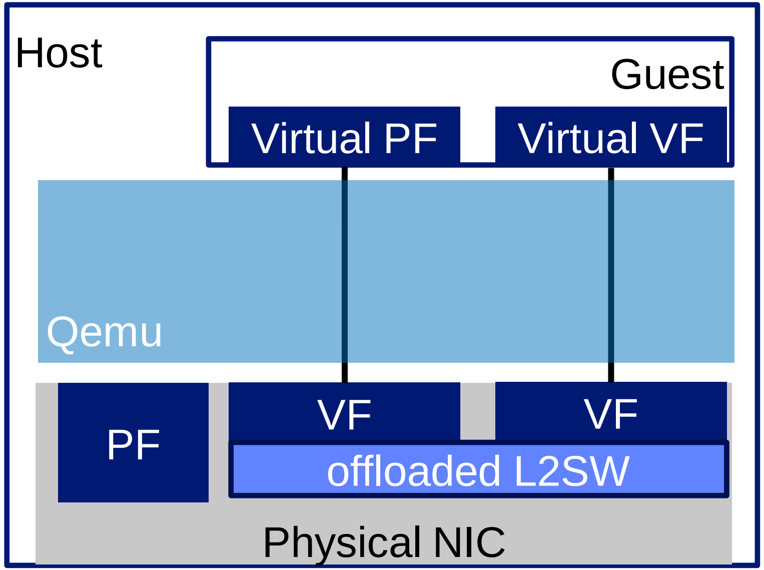
Utilize virtio data acceleration (vDPA) for backend

- offload only data plane without offloading control plane

Control Plane



Data Plane

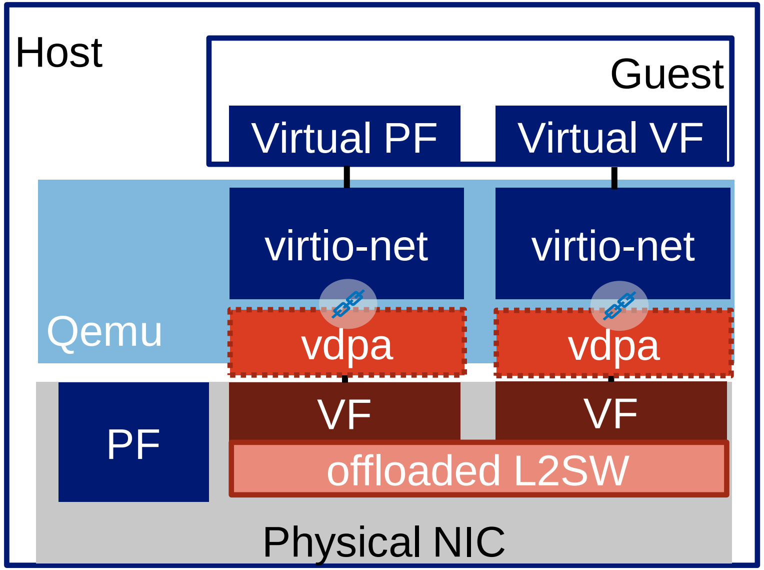


Useful for purpose other than acceleration

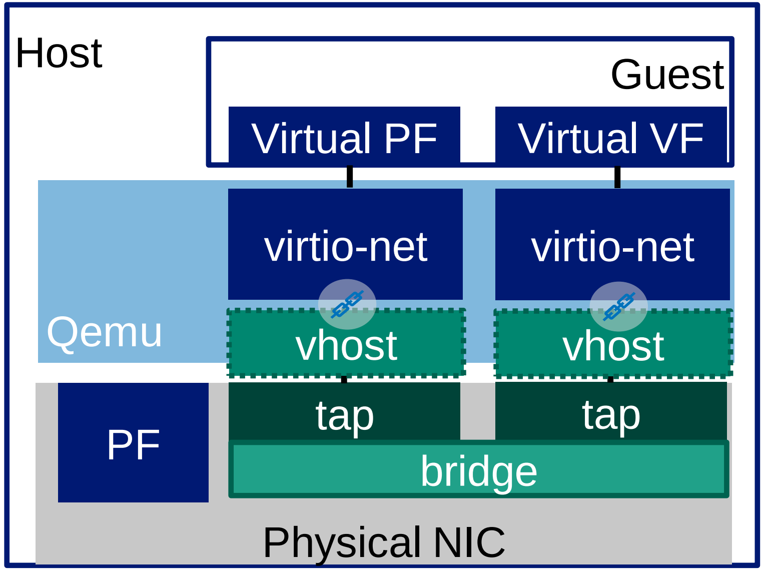
Replacing backend for other use cases

- Even utilize tap devices to test SR-IOV for guest OS as backend is pluggable

For acceleration



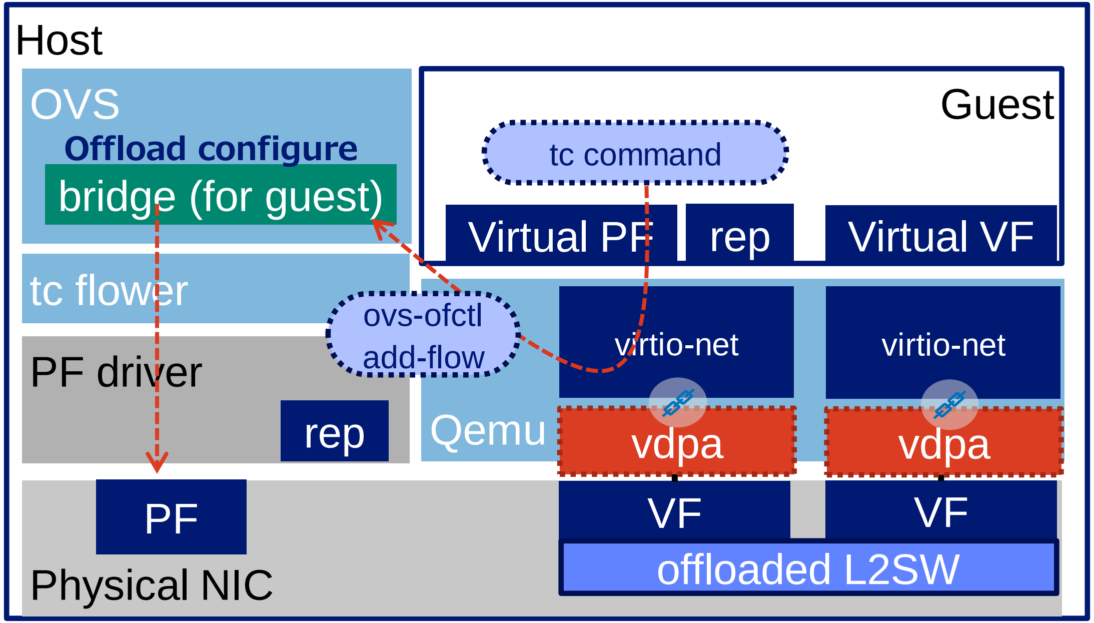
For testing & debugging



Future works: switchdev SR-IOV in guest

Potential solution for guest switchdev mode with OVS

- Qemu (+ libvirt etc.) handle tc configuration from guest
- Host's OVS configures offloading
- Bridge in OVS
  - For offloading guest's switch



Functional/Performance Verification

Unleashing SR-IOV Offload on Virtual Machines

Purpose of Verification

Confirm the followings:

- performance is improved by vDPA
- SR-IOV CNI plug-in works

Verification Target’s 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
Host/Guest kernel	6.5.7
Qemu version	Qemu 8.1.1 (w/ virtio-net legacy SR-IOV support PoC 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 (caused by current virtio limitation)
structure of virtio’s sysfs is different from other common devices because virtio bus is under virtio PCI device

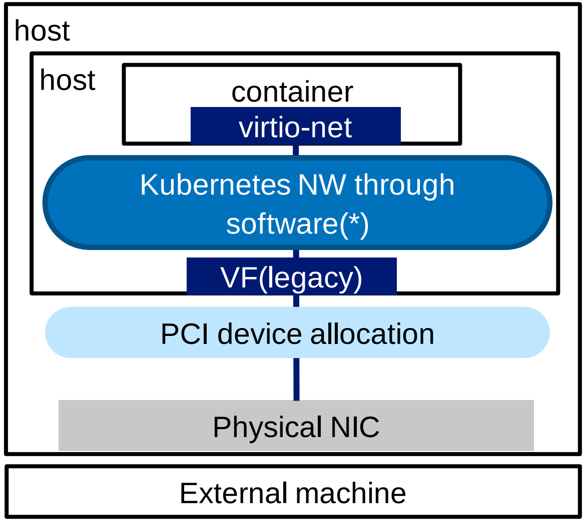
Target Environments

Functional/performance verification in the following 2 environments

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

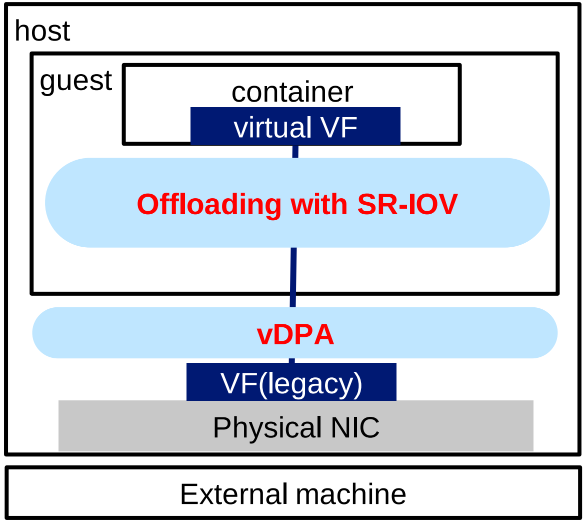
Without offload in VM

- Comparison target
- without using HW offload on guest



With offload in VM

- HW offload usage on guests by SR-IOV CNI

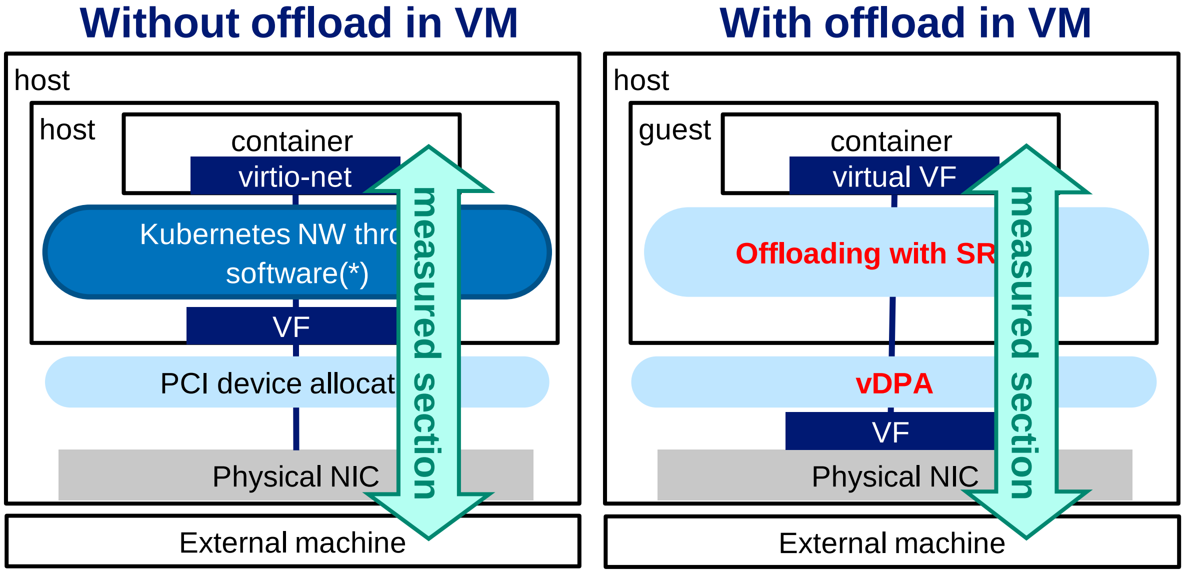


(*) including firewall and NAT

Verification metrics and section

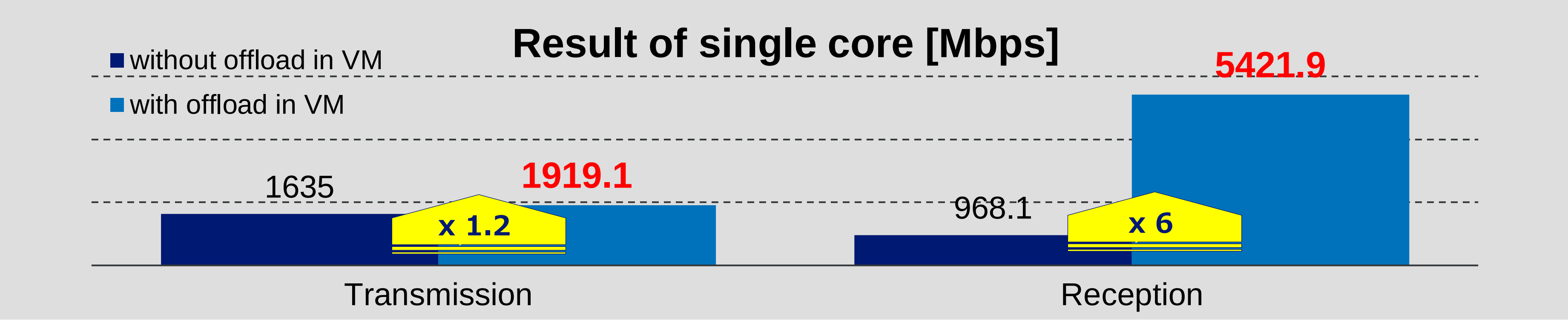
Verification metrics are throughput and latency

- Measured between application on guest's container and application on external machine



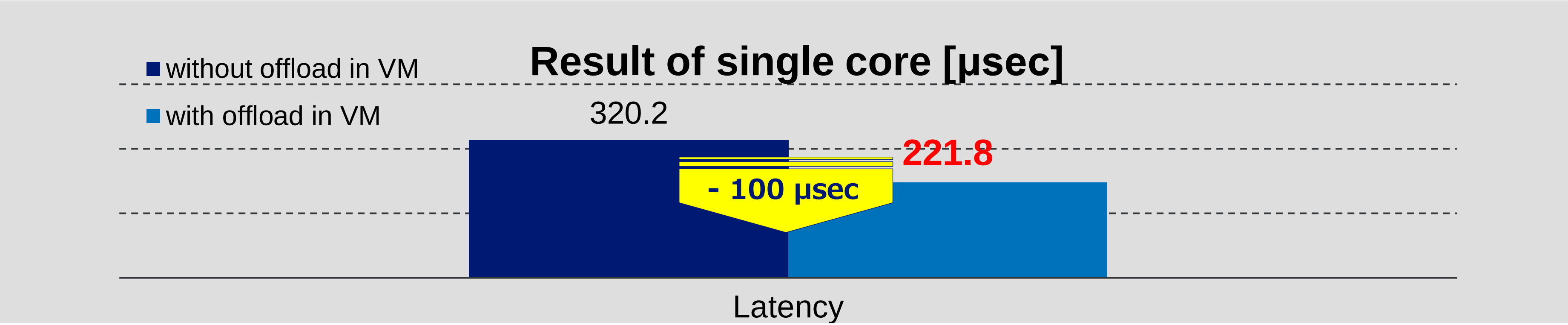
Throughput

- Measuring method
  - Average throughput of UDP bulk transfer using netperf
- Results
  - Transmission: **x 1.2** (1635 Mbps → 1919.1 Mbps)
  - Reception: **x 6** (968.1 Mbps → 5421.9 Mbps)



Latency

- Measuring method
  - 99%ile of round trip time: UDP request/response using netperf
- Results
  - **- 100 µsec** (320 µsec → 221 µsec)



Summary

Summary

- Background
  - There are cases: deploying containers to VMs for high flexibility
  - Achieve offloading Linux network on VMs to SR-IOV physical NICs
- Proposed method: Virtual PF
  - Application works same as on physical machine using SR-IOV emulation
  - Able to offload to NIC container’s virtual NW using vDPA
- Result
  - Throughput (Tx): **x 1.2** (1600 Mbps → 2800 Mbps)
  - Throughput (Rx): **x 6** (930 Mbps → 5500 Mbps)
  - Latency: **- 100 μsec** (320 μsec → 221 μsec)