Cloud and Datacenter Networking

Università degli Studi di Napoli Federico II
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione DIETI
Laurea Magistrale in Ingegneria Informatica

Prof. Roberto Canonico

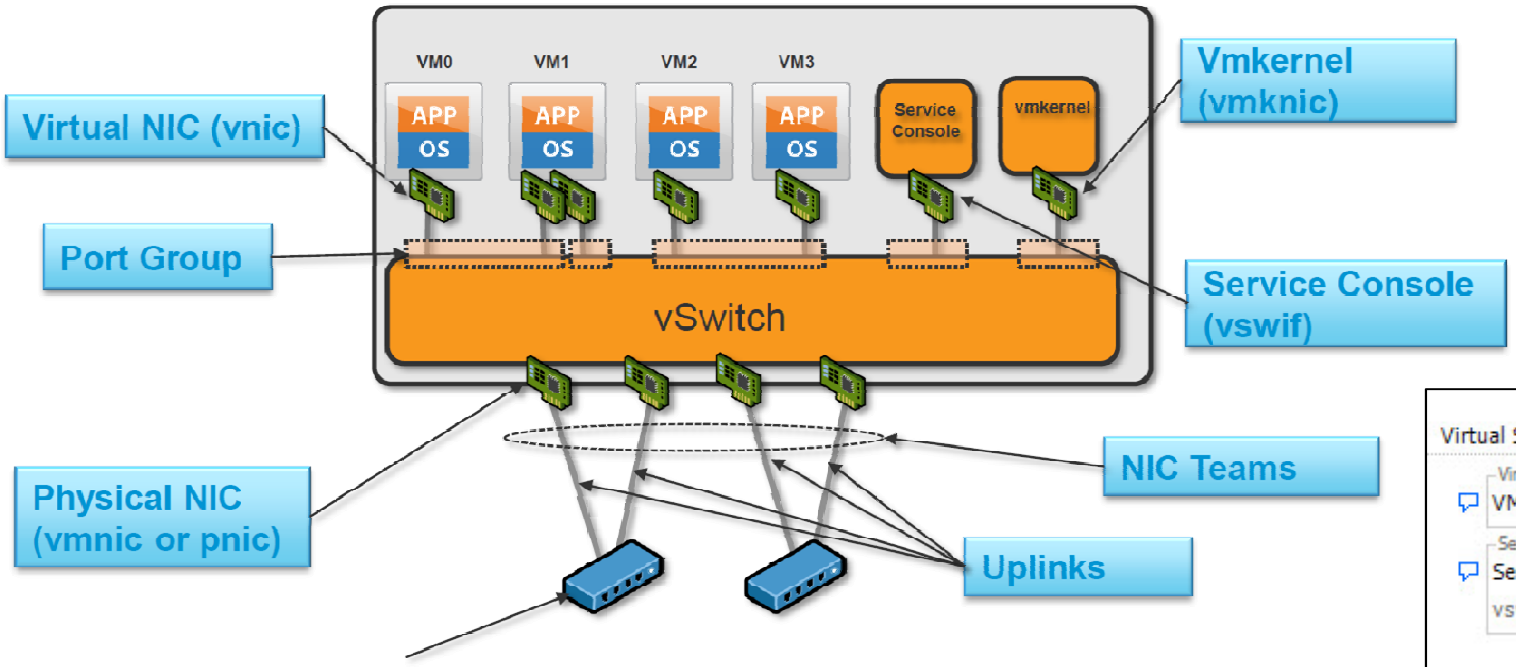
Virtual Machine communication mechanisms
Virtual switches – Open vSwitch

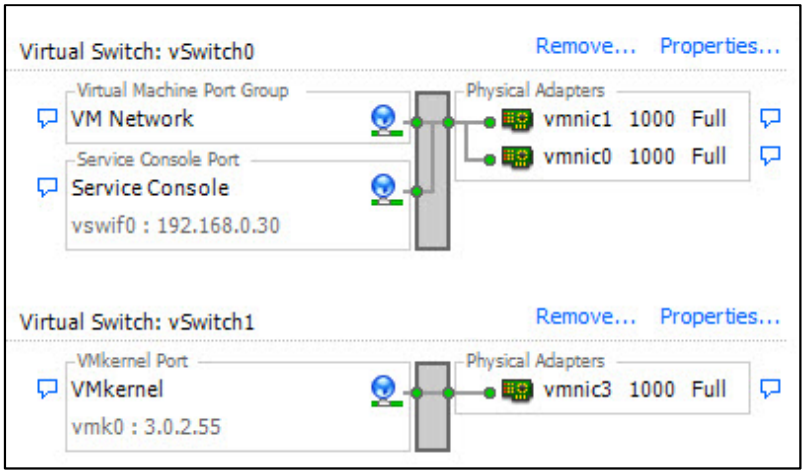
Lesson outline

- Virtual networking techniques for VMs
- Software virtual switches in Linux-based hypervisors
- Macvlan
- Linux Bridge
- Open vSwitch
- Hardware support to virtual networking in modern NICs: SR-IOV

Hypervisors and VM networking (1)

- In a physical host with several VMs, each VM has it own virtual NIC(s) (or vNICs)
- Virtual NICs are connected to the host physical NIC(s) by means of a virtual switch (or vSwitch) whose job is to dispatch packets from/to VMs according to their virtual MAC addresses
- A single hypervisor may be configured with several vSwitches





Hypervisors and VM networking (2)

- A hypervisor creates virtualized network devices: vNICs and vSwitches
- Many VMs connected by virtual network devices create a *virtual network*
- Virtual network interface card (vNIC)
  - Hypervisor can create one or more vNICs for each VM
  - The vNIC provides the networking capabilities of the VM
  - Each vNIC is identical to a physical NIC
- Virtual switch(vSwitch)
  - Switches also can be virtualized as a virtual switch
  - Each vNIC is connected to a vSwitch port
  - A vSwitch may be connected to an external physical network through a physical NIC (pNIC) of the hypervisor

VMware terminology

- Virtual Network Adapters
  - vNic – VM's interface to the network
  - vmknic – vSphere hypervisor's interface to network (nfs, iSCSI, vMotion, FT, Management)
  - vswif – Interface for Service Console (not present on ESXi)
- Physical Network Adapter
  - pNic – for communicating with entities outside ESX/ESXi host
- Virtual Switch
  - vSwitch – forwards packets between vNics, vmknics, and pNics
- Port Group
  - Group of ports sharing the same configuration (e.g. vlan)
- Uplinks: connections to physical switches
- NIC Team: a group of pNics connected to the same physical network

VMware: 3 types of Virtual Switches

- vNetwork Standard Switch (vSS)
  - Created and managed on a per-host basis
  - Support basic features such as VLAN, NIC teaming, port security
- vNetwork Distributed Switch (vDS)
  - Created and managed at vSphere vCenter
  - Supports all vSS features and more (PVLAN, traffic management, etc.)
  - NOTE: vSS/vDS share same etherswitch module, only control path differ
- Cisco Nexus 1000v (N1K)
  - Created and managed by VSM (either VM or hardware/Nexus 1010)
  - Supports features typically available in Cisco hardware switches

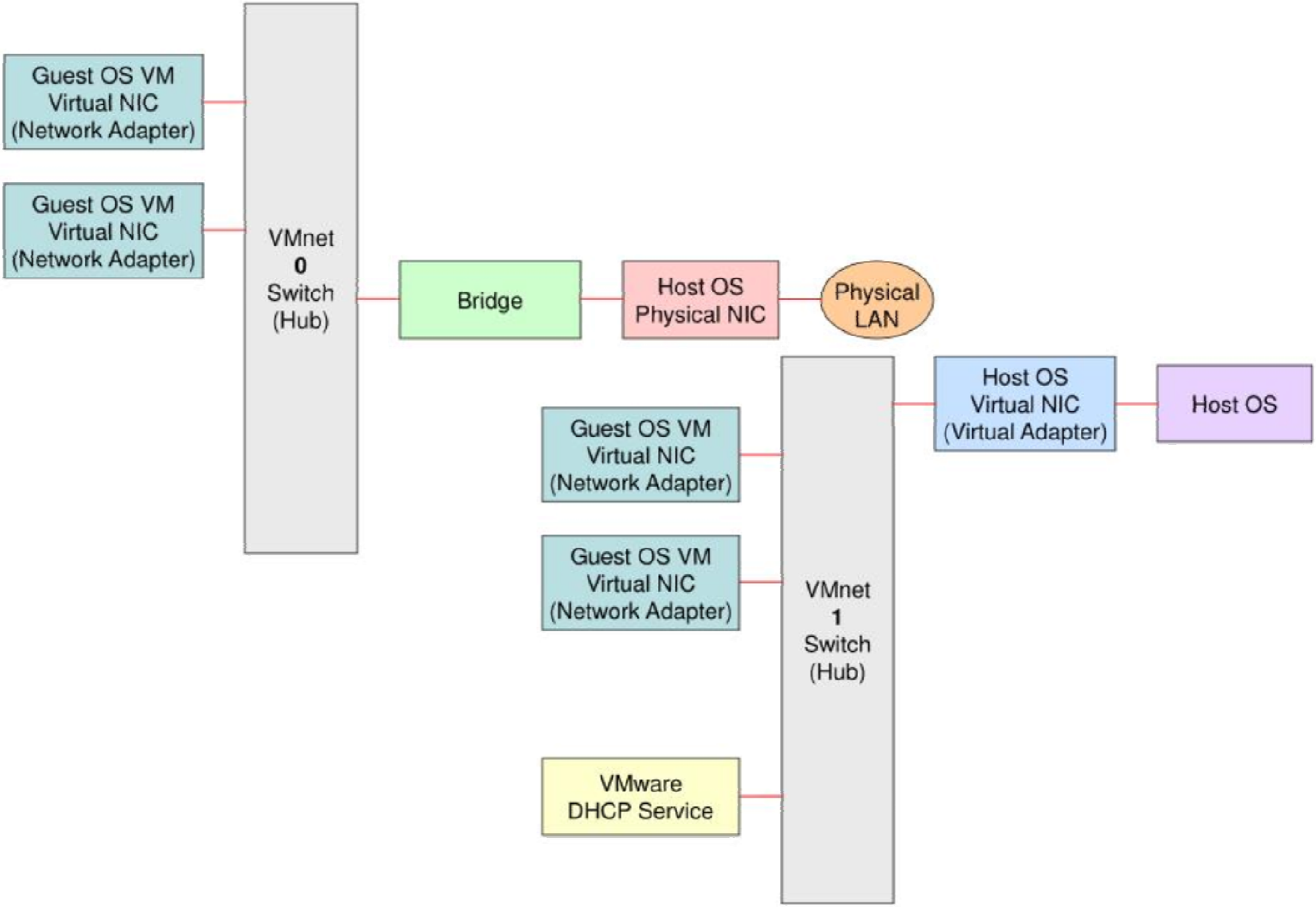
Networking modes in type-2 hosted hypervisors

- In type-2 hosted hypervisors, a VM virtual NIC may be configured in one the following three basic modes:
- **Bridged**: under the Bridged method, the VM will directly contact the DHCP server of the external physical network and apply for a unique local IP address in the external network; the VM will be then able to directly access the external network; this is the preferred connection method, if we run any server in the VM
- **NAT (Network Address Translation)**: under this method, the VM accesses the host's external network with the IP address of the host; within the host, we have a virtual private network involving the host and the VMs running on it. The other hosts in the external network cannot directly access the VM and they have to go through the NAT process at the host; in other words, the host PC acts as the first-stop gateway router for the VM
- **Host-only**: This method is same as the NAT except the VMs cannot access the external network as the host does not act as a NAT router for the VMs; communication is only allowed among VMs running in the same host

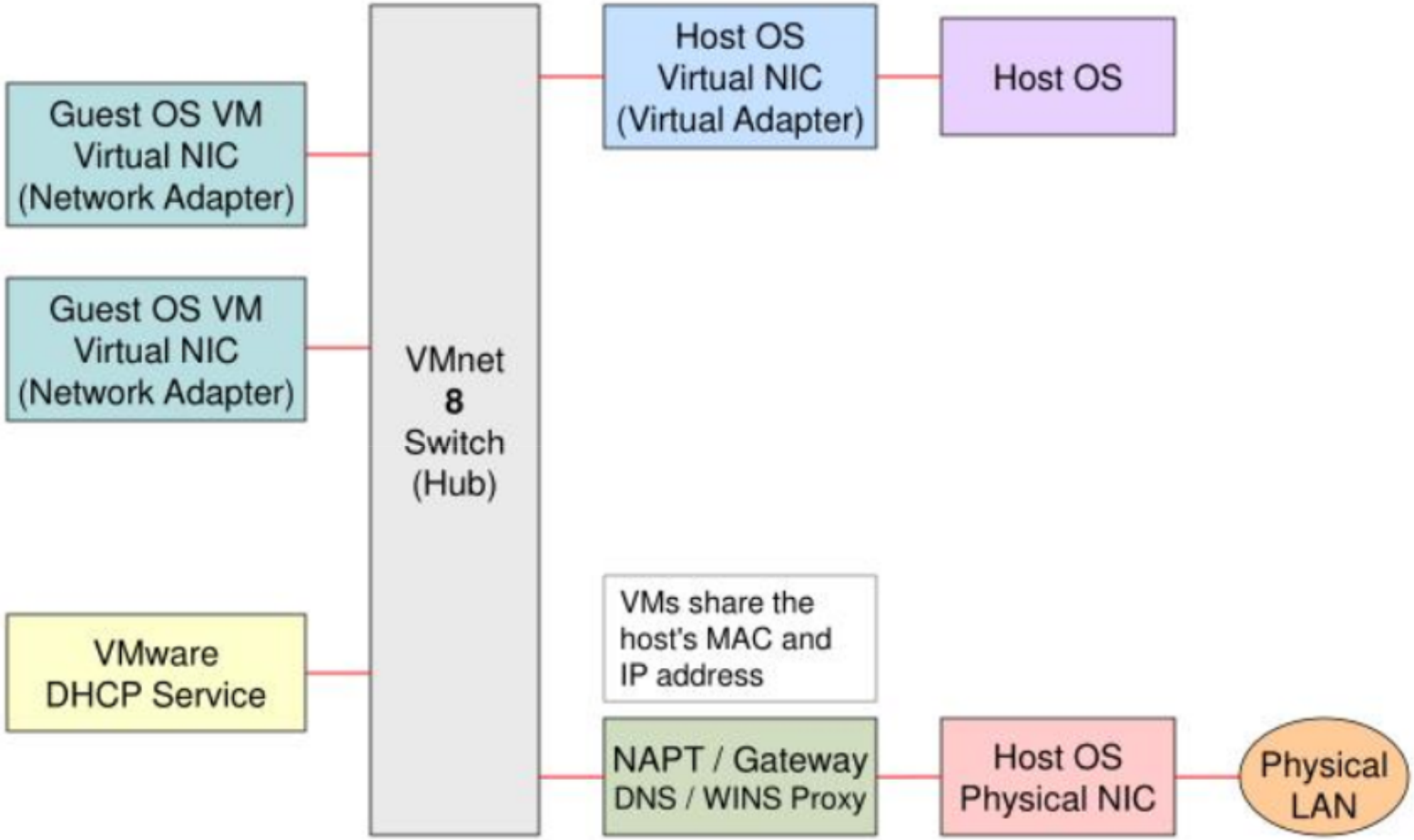
VMware hosted hypervisor: predefined virtual switches

- VMware produces different type-2 hypervisors
  - VMware Player (free for personal use)
  - VMware Player Pro
  - VMware Workstation
  - VMware Fusion
- VMware Workstation supports 10 virtual switches
  - Each virtual switch defines a virtual network: VMnet 0 – VMnet 9
- Networks 0, 1, and 8 are pre-configured
  - VMnet 0 is bridged
    - it is directly connected to the host external network through a host's physical NIC
  - VMnet 1 is host-only
    - it is connected to a virtual NIC which is added to the host;
    - a DHCP server is provided in this network by VMware itself
  - VMnet 8 is NATted
    - it is connected to a virtual NIC which is added to the host;
    - a DHCP server is provided in this network by VMware itself;
    - a NAT service forwards packets from VMs to the external network
- Networks 2, 3, 4, 5, 6, 7 and 9 are isolated networks by default

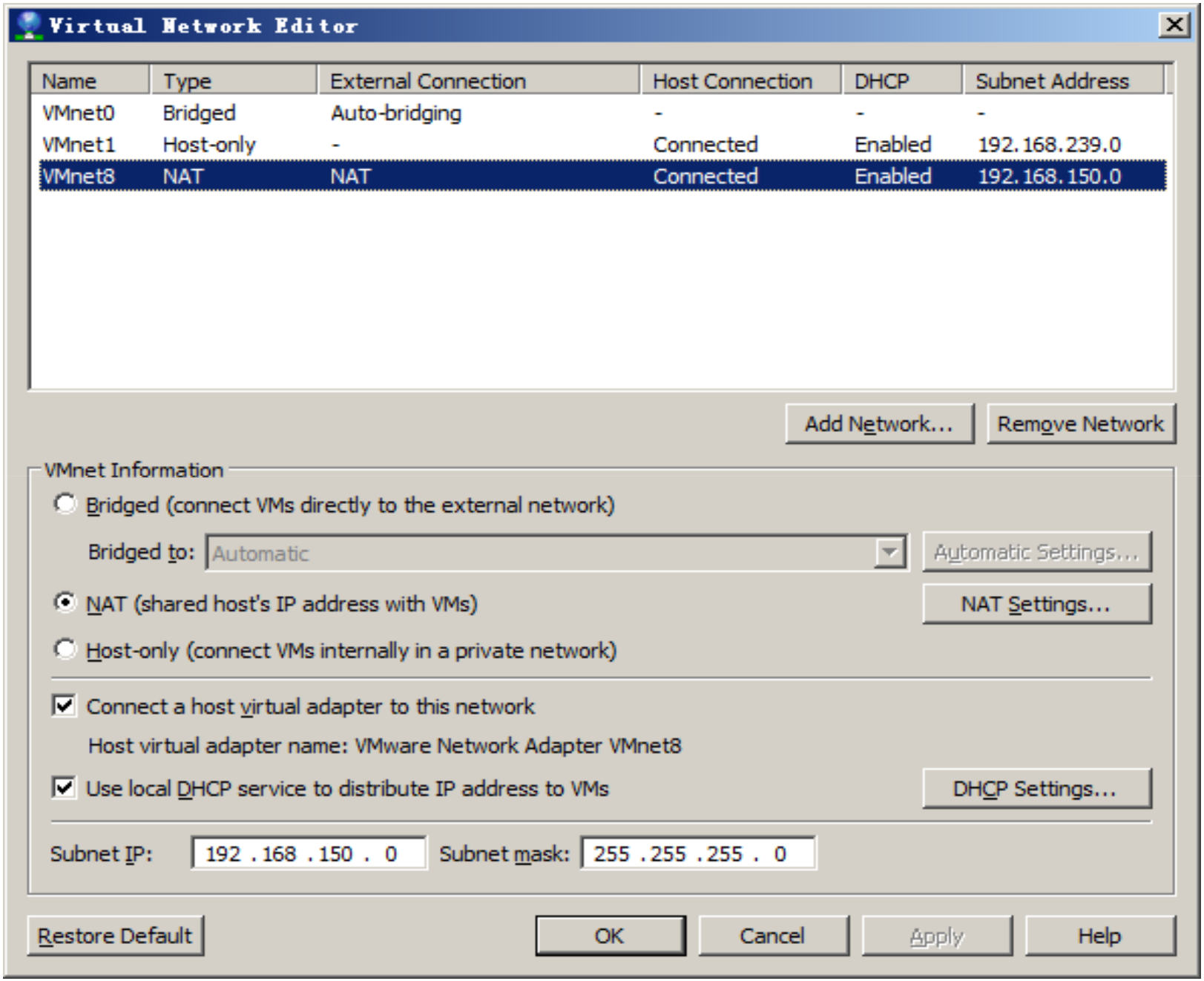
VMware hosted hypervisor: VMnet0 and VMnet1



VMware hosted hypervisor: VMnet8



VMware hosted hypervisor: Virtual Network Editor



Linux: 3 types of virtual switches

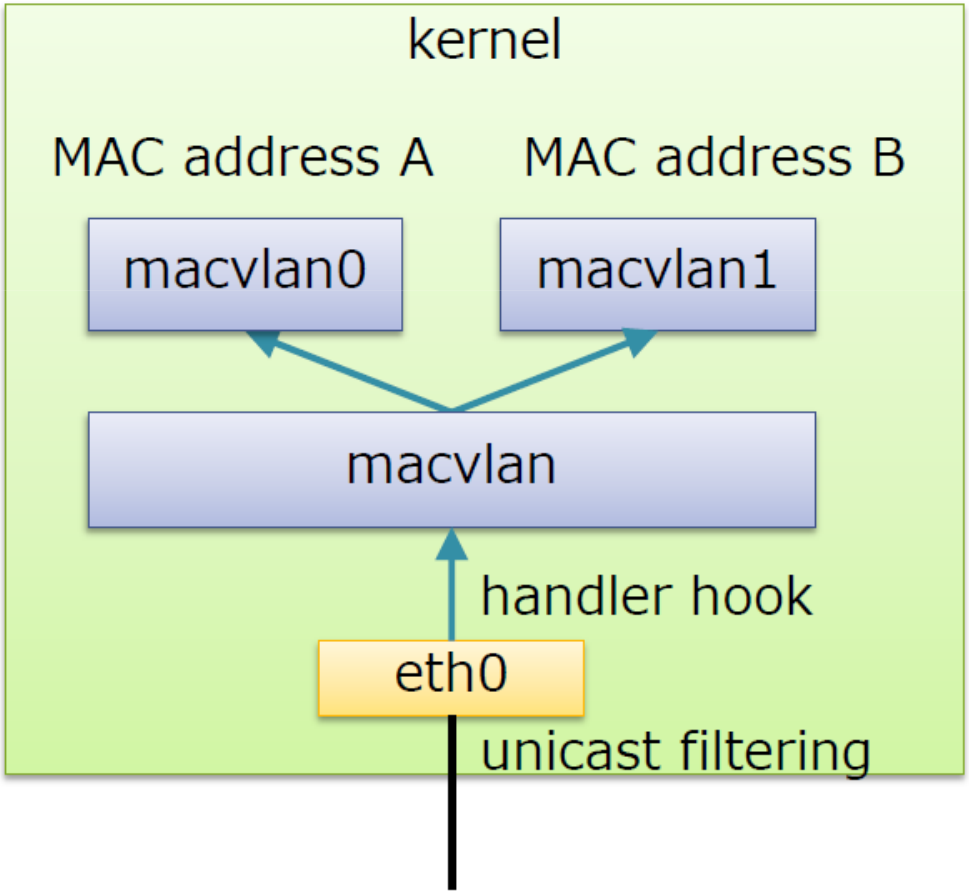
- Macvlan
- Linux Bridge
- Open vSwitch

Challenge for (virtual) switches: MAC table explosion

- Since MAC addresses are flat, there is no easy way to “group” multiple entries in a MAC filtering table (as for IP routes)
- Each learning switch has to keep the MAC addresses of all the VMs in the network

macvlan

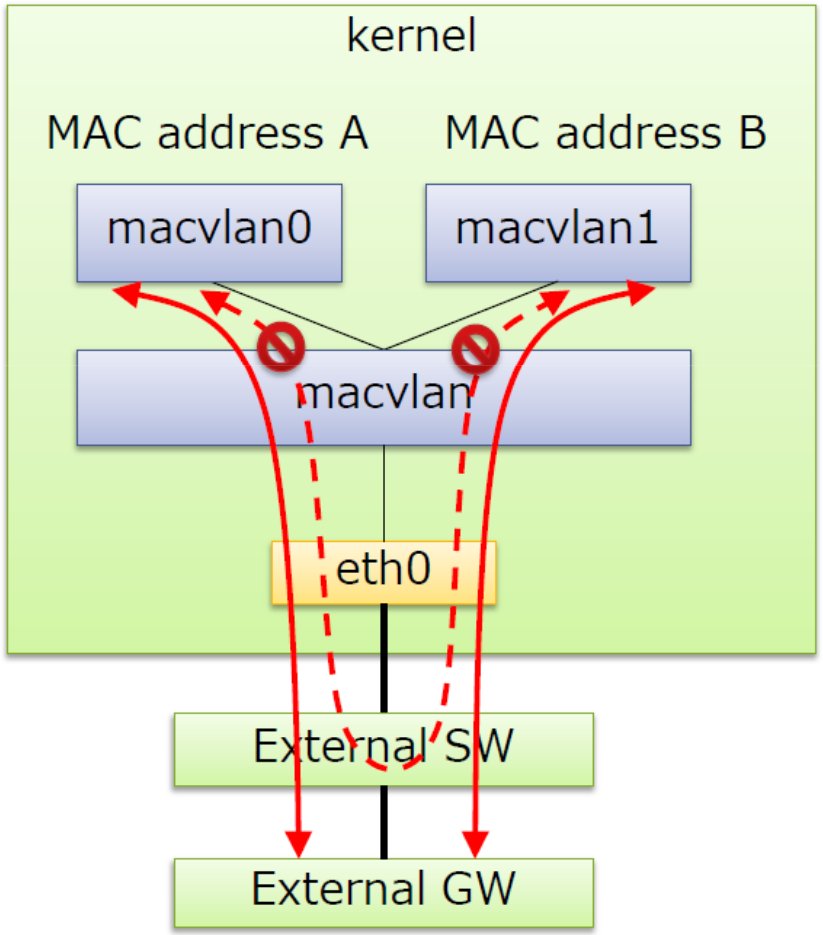
- With macvlan it is possible to configure a physical Ethernet NIC (*parent device*) with multiple sub-interfaces (*slave devices*), each with its own unique (randomly generated) MAC address
- macvlan creates different VLANs in which membership is associated to MAC addresses of VM virtual NICs
- Packets are not tagged (802.1q)
- If the physical NIC allows filtering of multiple unicast MAC addresses, this feature is used instead of promiscuos mode
- Four modes of operation:
  - Private
  - Vepa
  - Bridged
  - Passthru



Toshiaki Makita. *Virtual switching technologies and Linux bridge.* NTT Open Source Software Center.

Macvlan: private mode

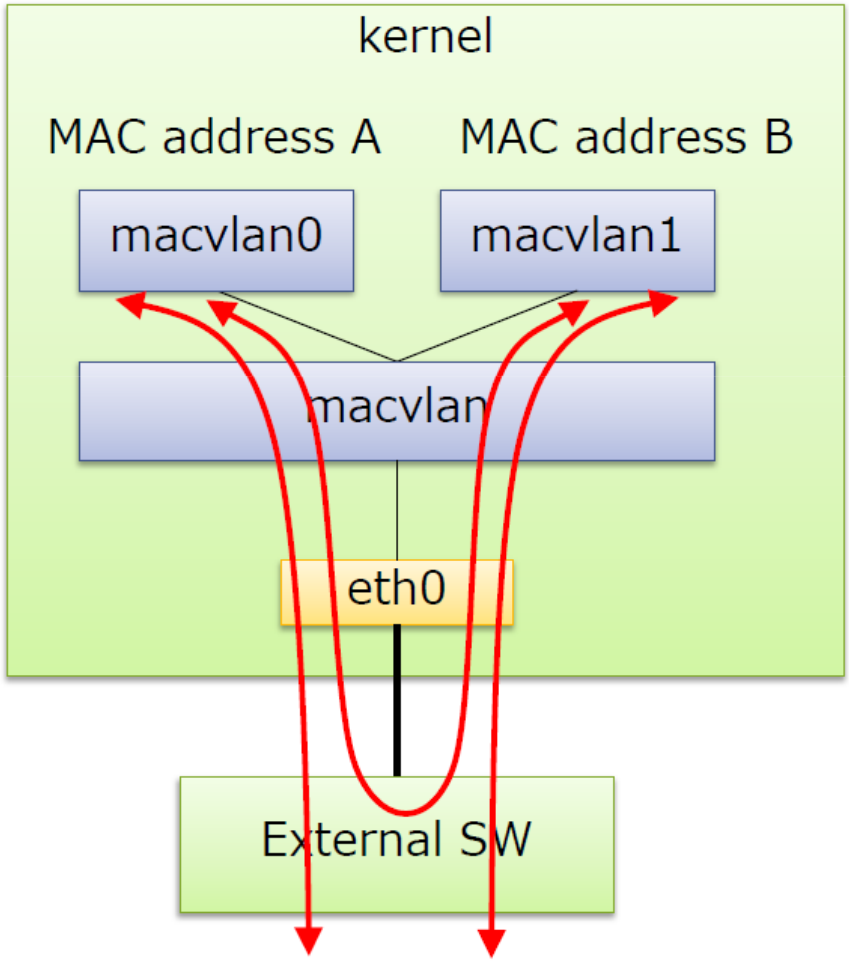
- In private mode the macvlan device does not forward packets among its ports
- Even if physical switch reflects the frame sourced from one sub-interface and destined to another sub-interface, frame gets dropped
- VM-to-VM communication needs an external gateway (routing) device
  - MAC addresses A and B belong to different VLANs



Toshiaki Makita. *Virtual switching technologies and Linux bridge*. NTT Open Source Software Center.

Macvlan: VEPA mode

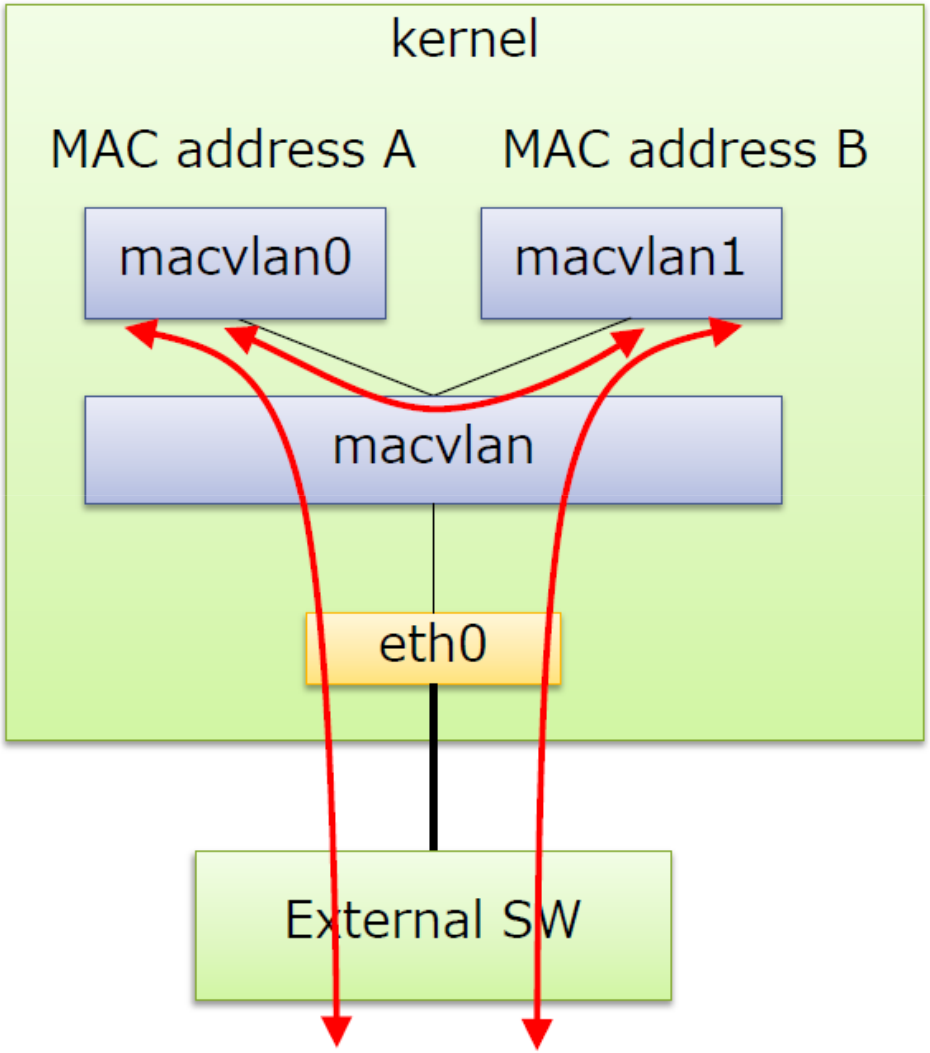
- In VEPA mode the macvlan device does not forward packets among its ports
- VM-to-VM communication may happen through an external switch
- VEPA mode requires an IEEE 802.1Qbg aka Virtual Ethernet Port Aggregator physical switch
- In VEPA mode, broadcast frames coming in through the parent interface get flooded to all macvlan interfaces
- VEPA mode is useful when policies are enforced on the physical switch and you want all VM-to-VM traffic to traverse the physical switch



Toshiaki Makita. *Virtual switching technologies and Linux bridge*. NTT Open Source Software Center.

Macvlan: bridge mode

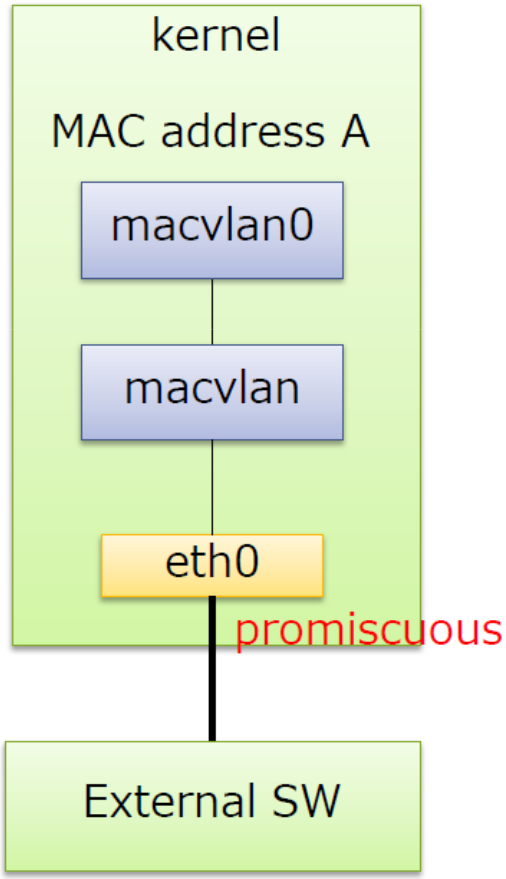
- In bridge mode the macvlan device is able to forward packets among its ports
- VM-to-VM communication may happen internally through macvlan device
- The macvlan device bridging capabilities are minimal
- Since all macvlan sub-interface MAC addresses are known, macvlan bridge mode does not require source MAC learning and does not need STP
  - Only one uplink



Toshiaki Makita. *Virtual switching technologies and Linux bridge*. NTT Open Source Software Center.

Macvlan: passthru mode

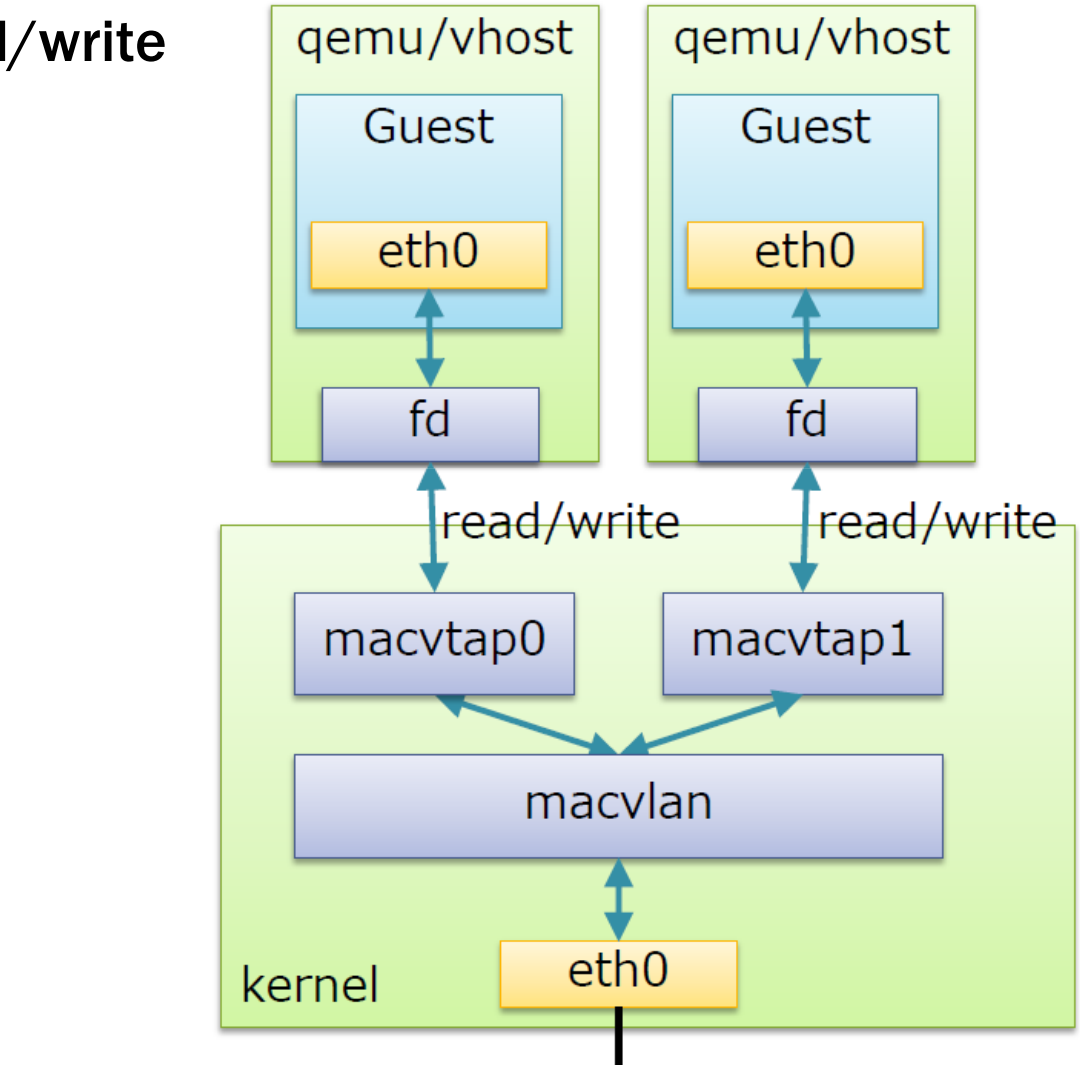
- In passthru mode only one virtual device per macvlan is allowed
- The physical NIC is put in promiscuos mode



Toshiaki Makita. *Virtual switching technologies and Linux bridge*. NTT Open Source Software Center.

KVM and macvlan with TAP interfaces

- VMs could be connected to the macvlan device not directly but through virtual TAP devices that expose a file read/write interface
- packet reception → file read
- packet transmission → file write



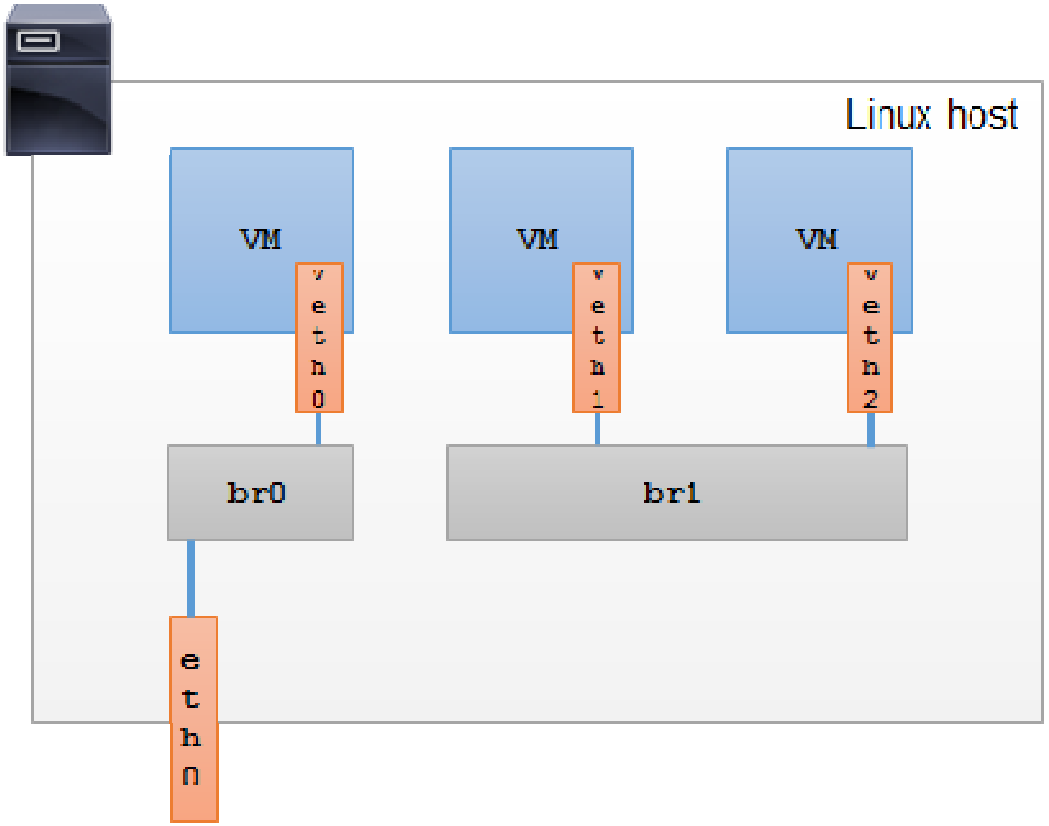
Toshiaki Makita. *Virtual switching technologies and Linux bridge*. NTT Open Source Software Center.

Linux Bridge

- Linux Bridge is a virtual network device working at layer 2 as an Ethernet physical switch
- A Linux Bridge can bind other Linux network device as a slave device, and virtualize the slave device as a port
  - using promiscuous mode that allows to receive all packets
- To install Linux Bridge in a Debian-based Linux distribution:

```
sudo apt-get install bridge-utils
```

- bridge-utilities is a program that implements a subset of the IEEE 802.1d standard and also comprises STP (*Spanning Tree Protocol*)
- bridge-utilities consists in
  - a Kernel module and
  - a user space application (*brctl*)



brctl basic commands

- Create/destroy a bridge device:

```
brctl addbr bridge_name
brctl delbr bridge_name
```

- Add/delete interface to a bridge device:

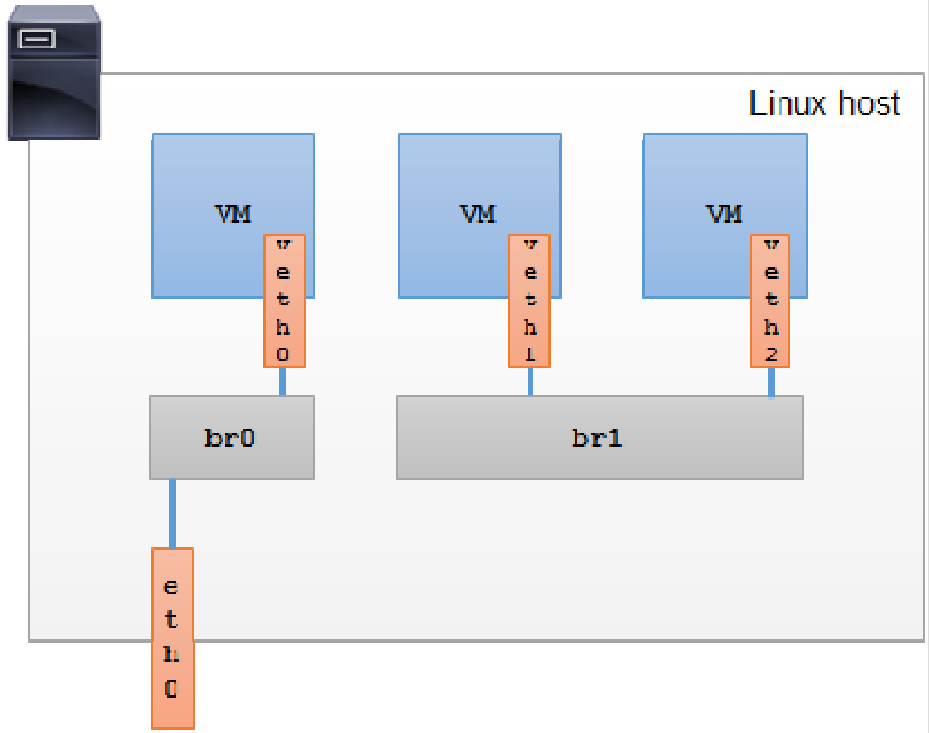
```
brctl addif bridge_name device_name
brctl delif bridge_name device_name
```

- Show devices in a bridge:

```
brctl show
```

- Show the forwarding DB:

```
brctl showmacs bridge_name
```

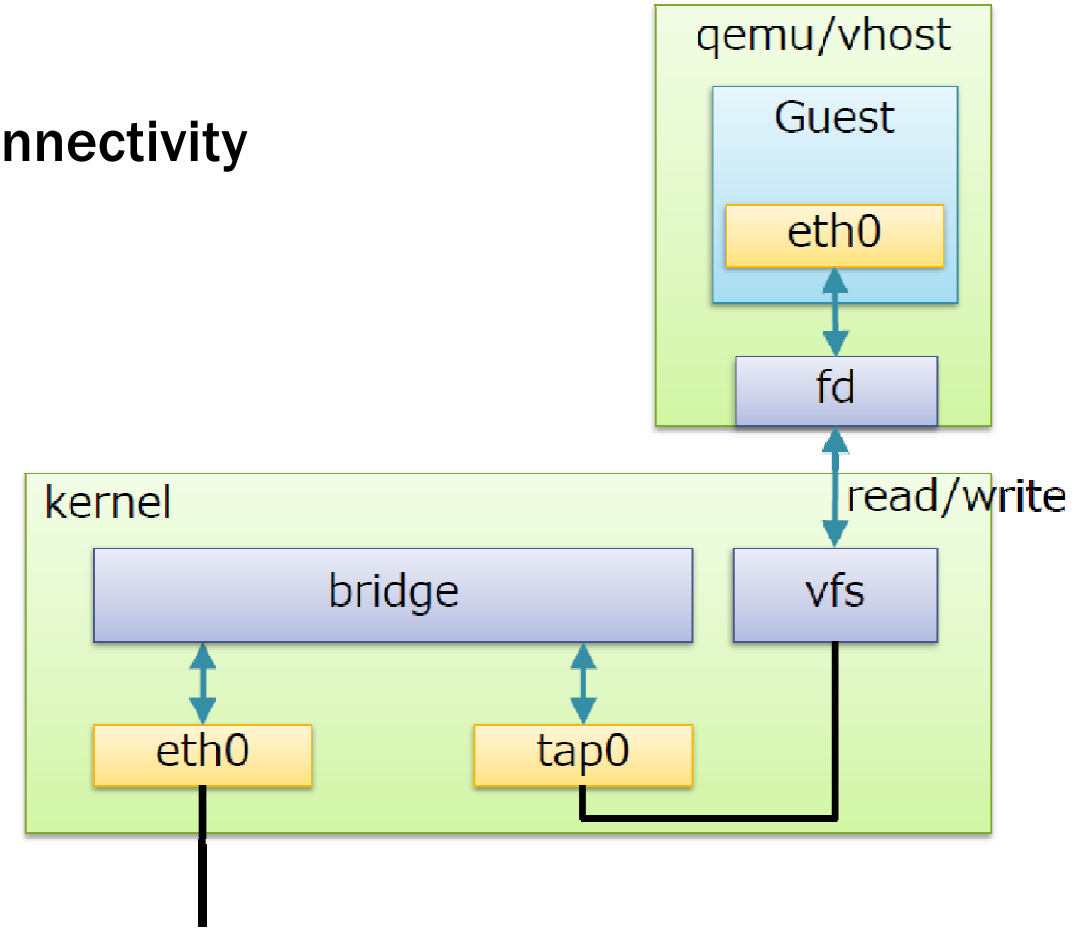


- Example of `brctl show` output:

```
bridge name     bridge id               STP enabled     interfaces
br0             8000.080006ad34d1       no              eth0
                                                        veth0
br1             8000.080021d2a187       no              veth1
                                                        veth2
```

KVM and Linux bridge with TAP interfaces

- VMs create an internal connection to a virtual TAP device
- The TAP device is configured as a port for the Linux Bridge
- VM-to-VM communication may happen through the Linux Bridge
- A physical NIC (e.g. *eth0*) provides connectivity towards the rest of the world



Toshiaki Makita. *Virtual switching technologies and Linux bridge.* NTT Open Source Software Center.

Tap devices in a KVM hypervisor

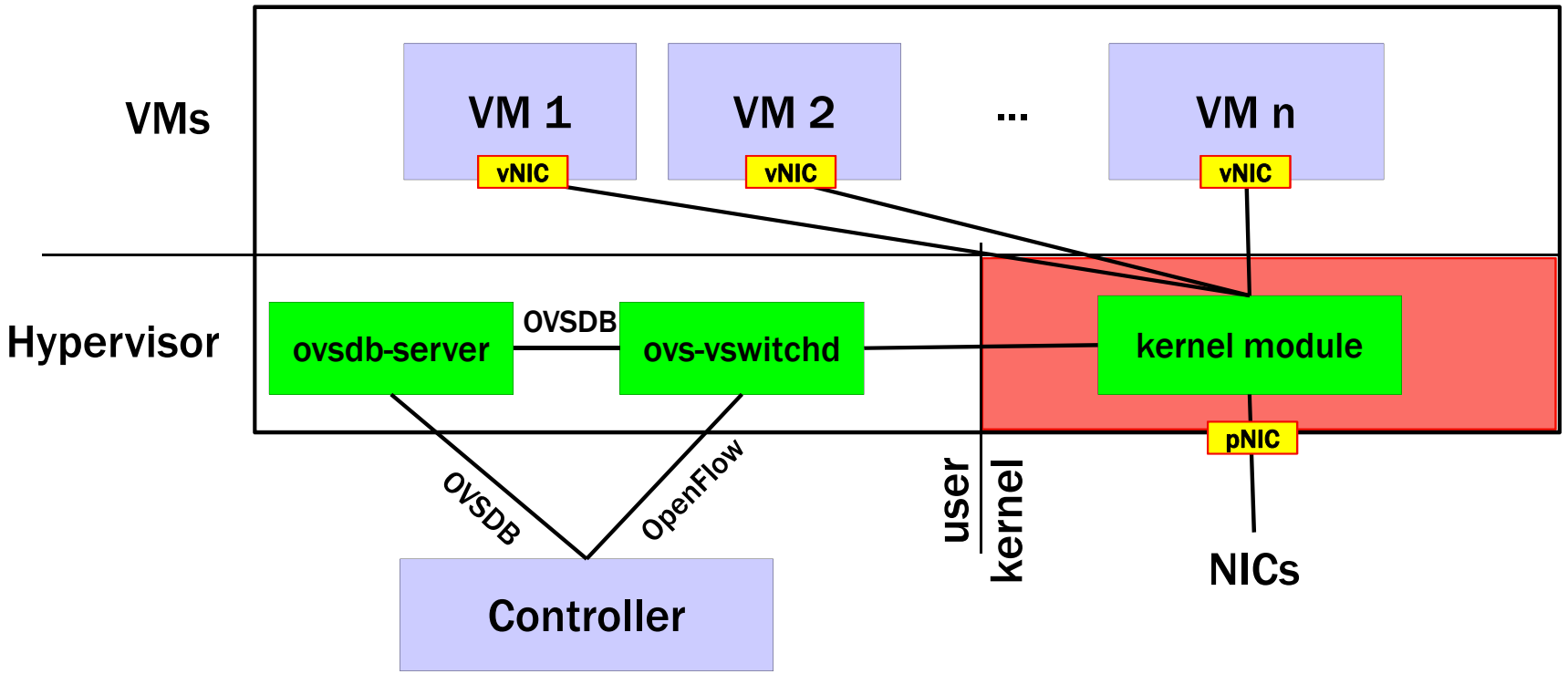
- Each VM corresponds to a tap device

```
[root@compute2 ~]# ifconfig |grep tap
tap0920d05a-38: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tap1c52b862-e1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tap2efedf47-39: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tap45787583-0b: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tap5140c9f1-87: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tap56b1fa1f-e5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tap69383260-61: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tap6bbe47a7-f2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tap7b92e41b-a3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tap8213fc6e-f9: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tap833640f0-24: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tap915224e1-da: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tap9bc4ac99-b9: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tapb8e468e6-59: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tapbb4b88b7-1f: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tapcaf75f1d-9b: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tapdaa7ce9a-7c: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tapdea66e2e-63: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
tape28c2f3a-a5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

Open vSwitch

- "Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag)."
- Key design decision of Open vSwitch is to partition functions among kernel and user space
  - Performance-limiting operations (*packet forwarding*) executed in kernel space
  - Control plane operations are executed in user space
- In kernel space, to speedup forwarding decisions, these are taken by calculating a hash function on the tuple (src-MAC, dst-MAC, dst-IP, dsp-TCP-port) and stored in a cache to keep forwarding decisions within kernel space
  - Only in case of a cache miss (fist packet of a flow), user-space classifiers executed
  - Subsequent packets of a flow match a cached rule

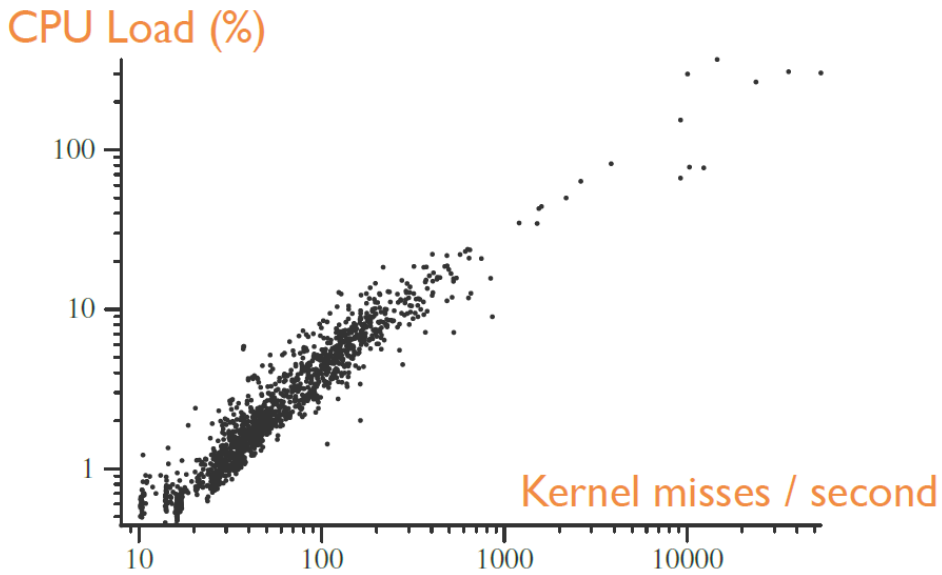
Open vSwitch Architecture



User space classifiers may be programmed and controlled by an external entity (OpenFlow controller)

Open vSwitch performance

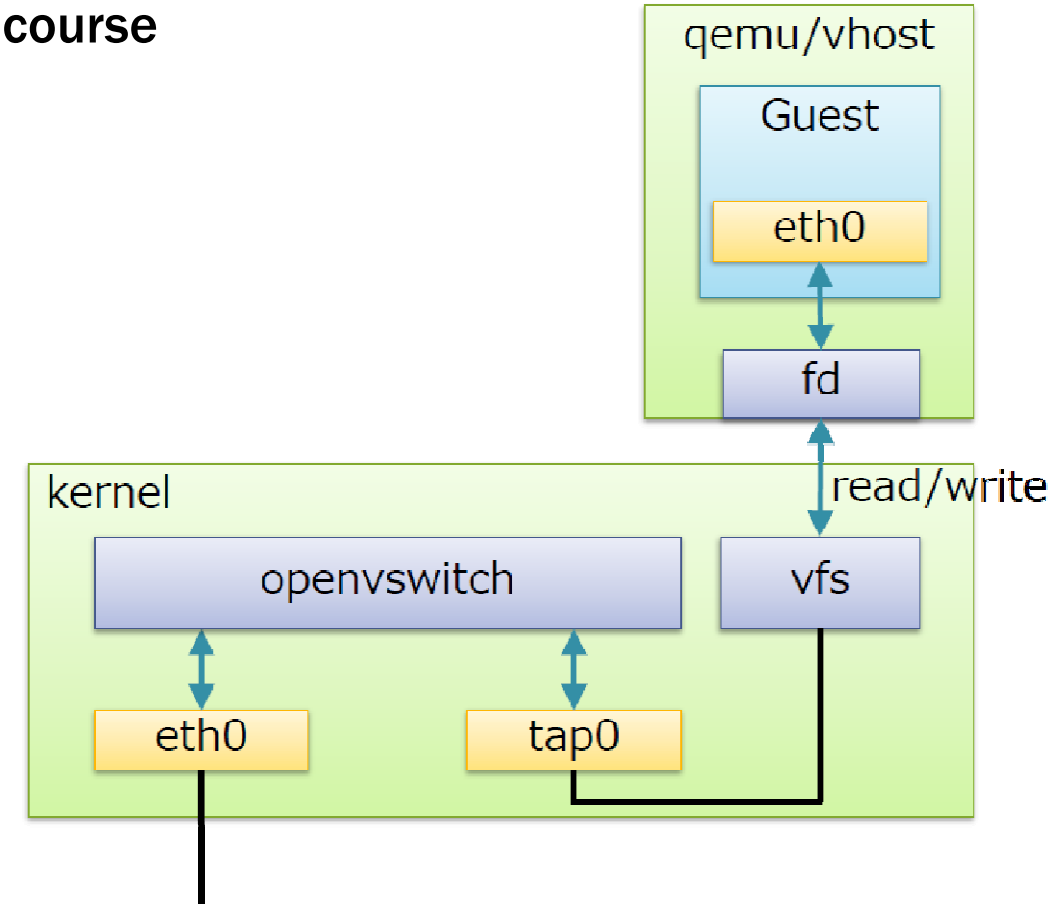
- Performance evaluation of Open vSwitch presented in [*] on a real deployment of a large number (> 1000) of hypervisor nodes dealing with huge traffic (24h)
- The more the caching function within the kernel is bypassed (cache miss), the more the host CPU is loaded
- However, for the vast majority of cases, CPU is never loaded more than 20%
- Over 80% of nodes with a CPU load less than 5%



[*] [The Design and Implementation of Open vSwitch. Pfaff et. al, USENIX NSDI 2015]

KVM and openvswitch with TAP interface

- Same configuration as for the Linux Bridge
- Plug-and-play replacement of Linux Bridge with greater flexibility
  - See OpenFlow later on in the course



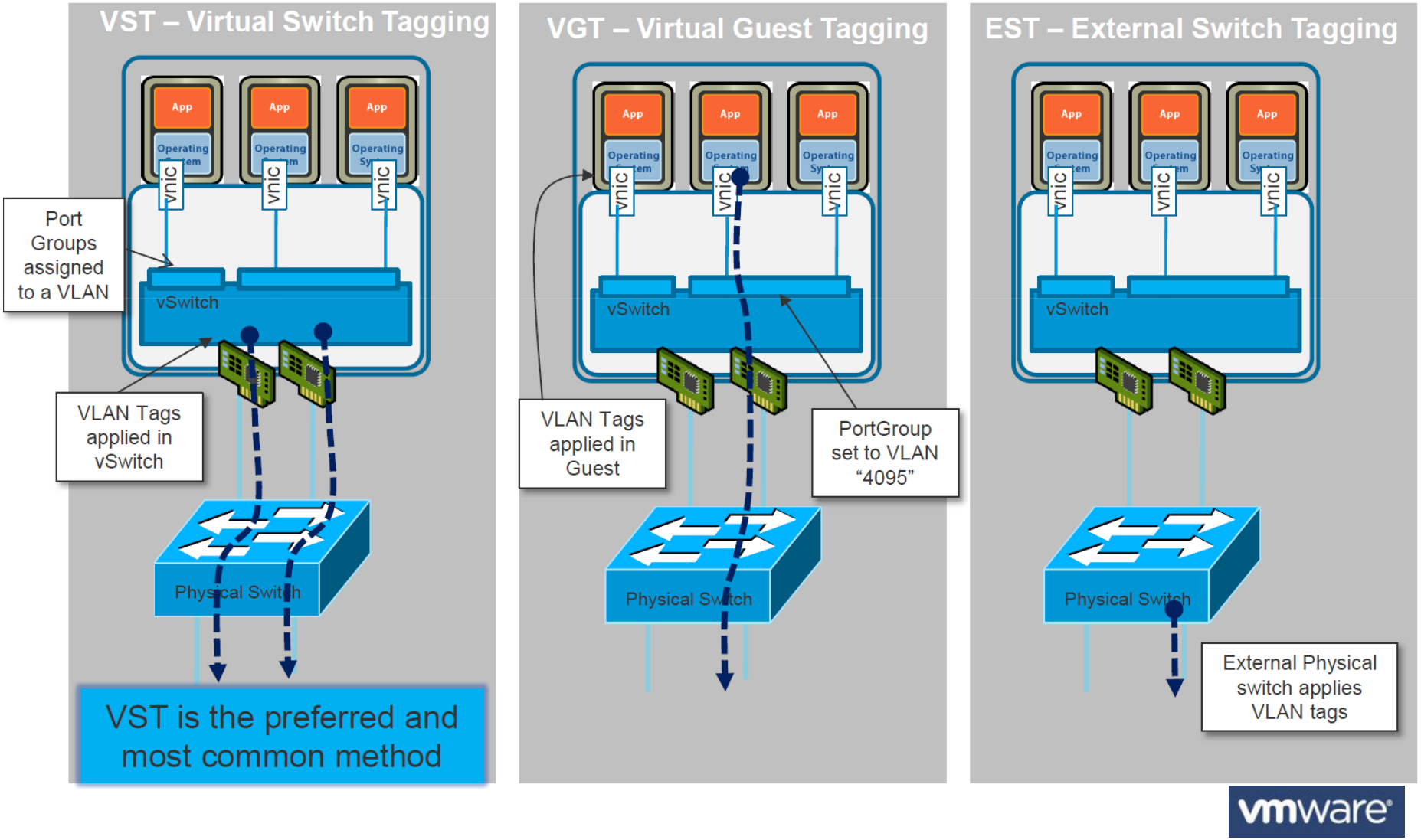
Toshiaki Makita. *Virtual switching technologies and Linux bridge*. NTT Open Source Software Center.

Open vSwitch vs Linux Bridge

Feature	Open vSwitch	Linux Bridge
MAC Learning Bridge	X	X
VLAN support (802.1Q)	X	Using ‘vlan’
Static Link Aggregation (LAG)	X	Using ‘ifenslave’
Dynamic Link Aggregation (LACP)	X	Using ‘ifenslave’
Support for MAC-in-IP encapsulation (GRE, VXLAN, …)	X	VXLAN support in 3.7 kernel + Iproute2
Traffic capturing / SPAN (RSPAN with encap. Into GRE)	X	Using advanced Traffic Control
Flow monitoring (NetFlow, sFlow, IPFIX, …)	X	Using ipt_netflow
External management interfaces (OpenFlow & OVSDB)	X	
Multiple-Table forwarding pipeline with flow-caching engine	X	

VLAN Tagging Options

- When a VM needs to participate in a VLAN spanning across several physical servers and switches, 802.1q VLAN tagging is needed
- Who tags packets ?

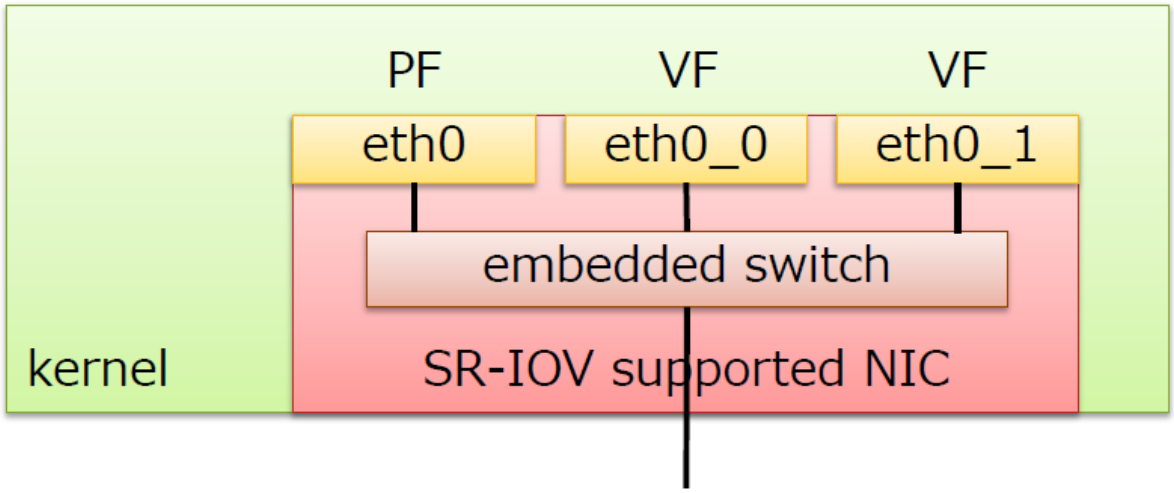


Virtual switches performance and SR-IOV

- Since a virtual switch requires some CPU processing to select the outgoing packet queue and to copy packets from one physical NIC to a virtual NIC (or viceversa), CPU power may limit the aggregate VM throughput
- At 10 Gbps, the time it takes to transmit an 84 bytes packet is 67 ns
- A single access to memory may require in the order of 10 ns
- To improve packet processing throughput in the case of many VMs, a new technology has been developed: SR-IOV (*Single Root I/O Virtualization*)
- SR-IOV relies on modern NICs

SR-IOV

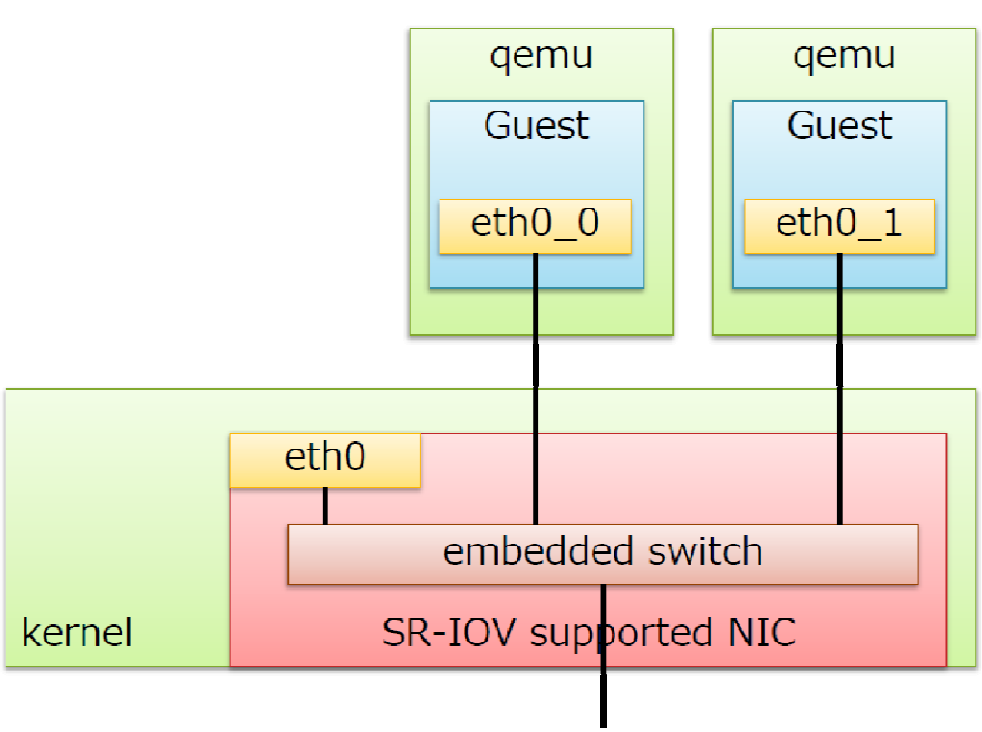
- Addition to PCI normal *physical function* (PF), allows to add lightweight *virtual functions* (VF)
- VF appears as a network interface
  - eth0_0, eth0_1, …
- SR-IOV devices have switches in them that allow PF-VF/VF-VF communication
- DMA is used to copy packets directly into VM's memory space without CPU load
- In terms of performance, SR-IOV produces higher throughput than software switches with much less CPU load

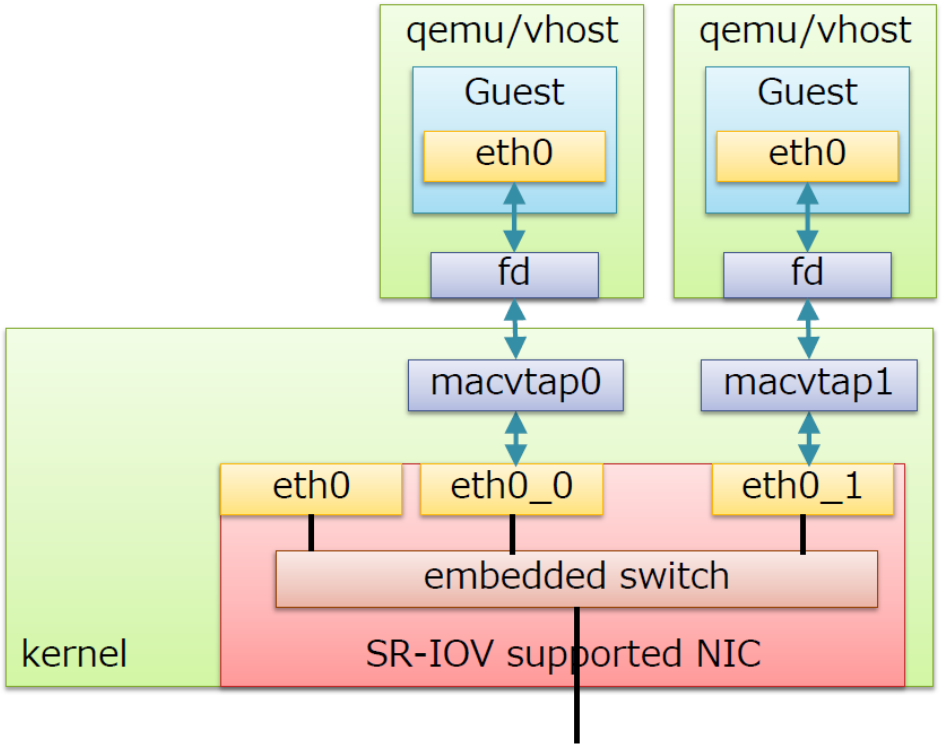


Toshiaki Makita. *Virtual switching technologies and Linux bridge*. NTT Open Source Software Center.

KVM with SR-IOV

- Two modes of operation:
  1. Use PCI-passthrough to attach VF to guest
  2. Use macvtap device (passthru)





Toshiaki Makita. ***Virtual switching technologies and Linux bridge.***
NTT Open Source Software Center.