Hardware Accelerating Linux Network Functions

Part I: Virtual Switching Technologies in Linux

Toshiaki Makita
NTT Open Source Software Center

Proceedings of netdev 0.1, Feb 14-17, 2015, Ottawa, On, Canada

Part I topics

- **Virtual switching technologies in Linux**
  - Software switches and NIC embedded switch
  - Userland APIs and commands for bridge
- **Introduction to Recent features of bridge (and others)**
  - FDB manipulation
  - VLAN filtering
  - Learning/flooding control
  - Non-promiscuous bridge
  - VLAN filtering for 802.1ad (Q-in-Q)
- **Demo**
  - Setting up non-promiscuous bridge

Who is Toshiaki Makita?

- **Linux kernel engineer at NTT Open Source Software Center**
- **Technical support for NTT group companies**
- **Active patch submitter on kernel networking subsystem**
  - bridge, vlan, etc.

Switching technologies in Linux

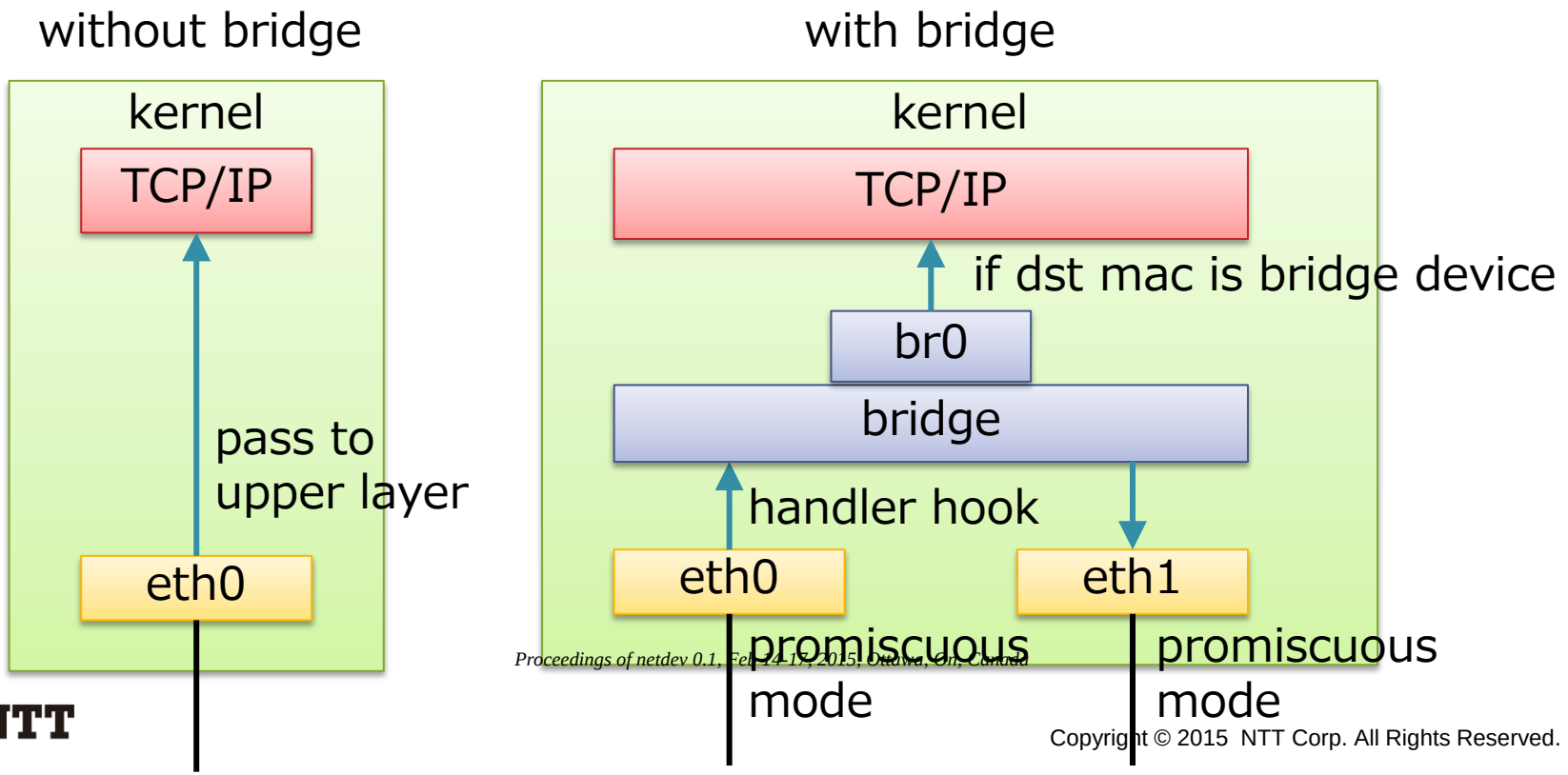
Innovative R&D by NTT

- **Linux (kernel) has 3 types of software switches**
  - bridge
  - macvlan
  - Open vSwitch
- **NIC embedded switch in SR-IOV device is also used instead of software switches**
- **These are often used for network backend in server virtualization**

NTT

bridge

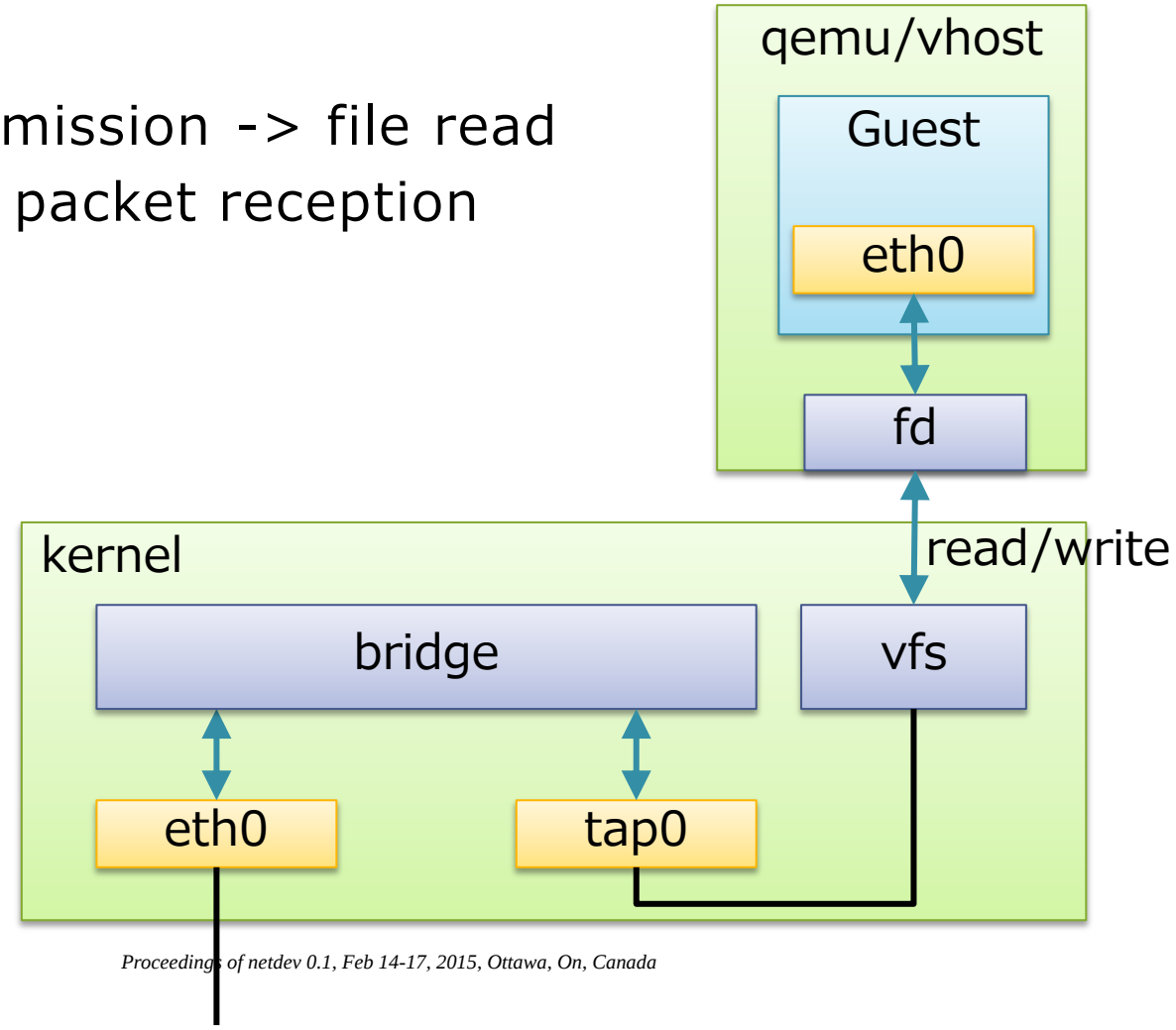
- **HW switch like device (IEEE 802.1D)**
  - Has FDB (Forwarding DB), STP (Spanning tree), etc.
  - Use promiscuous mode that allows to receive all packets
    - Common NICs filter unicast whose dst is not its mac address without promiscuous mode
    - Many NICs also filter multicast / vlan-tagged packets by default



Proceedings of netdev 0.1, Feb 14-17, 2015, Ottawa, On, Canada

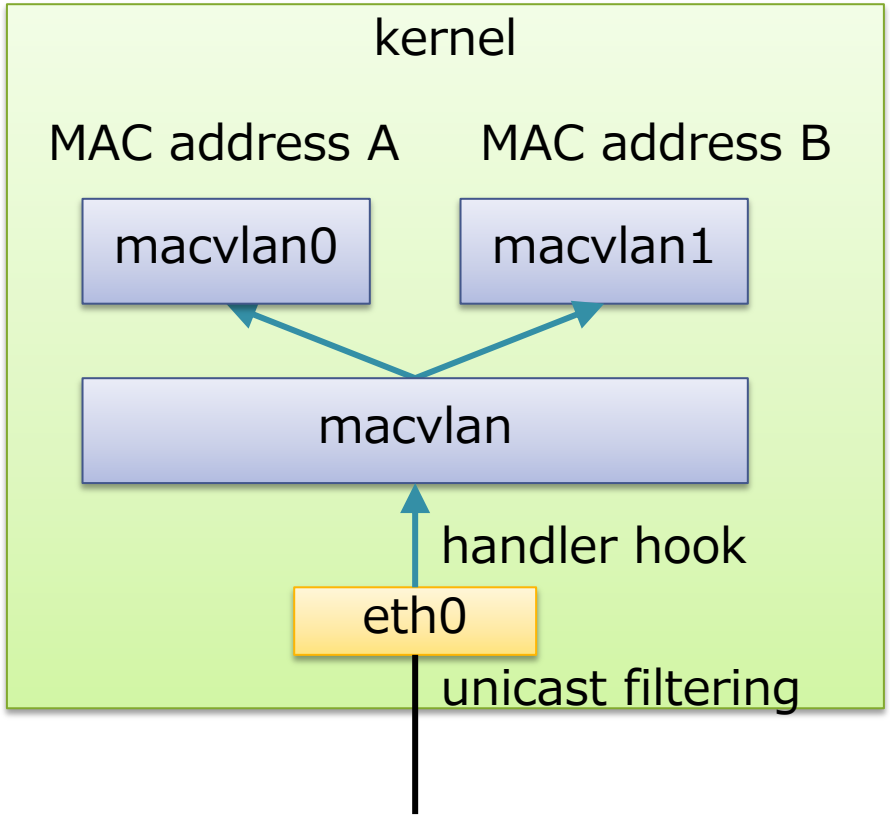
bridge with KVM

- **Used with tap device**
- **Tap device**
  - packet transmission -> file read
  - file write -> packet reception



macvlan

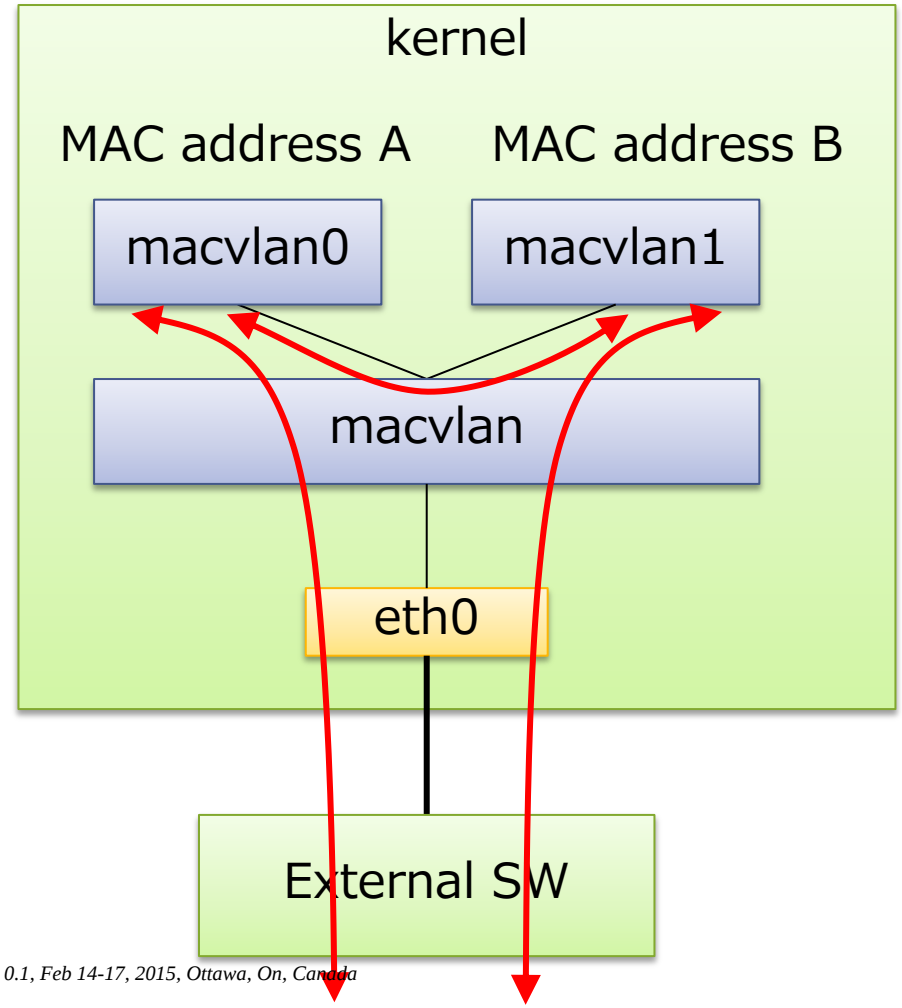
- **VLAN using not 802.1Q tag but mac address**
- **4 types of mode**
  - private
  - vepa
  - bridge
  - passthru
- **Using unicast filtering if supported, instead of promiscuous mode (except for passthru)**
  - Unicast filtering allows NIC to receive multiple mac addresses



Proceedings of netdev 0.1, Feb 14-17, 2015, Ottawa, On, Canada

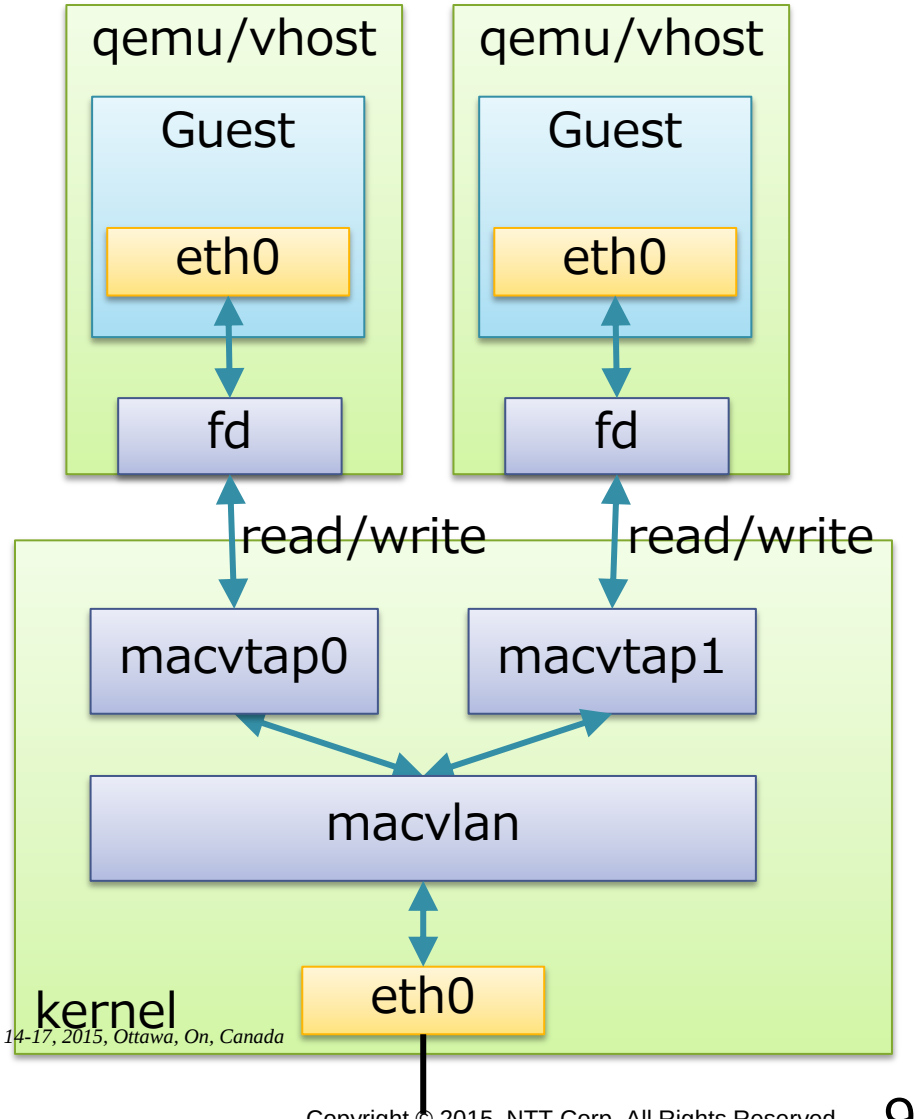
macvlan (bridge mode)

- **Light weight bridge**
  - No source learning
  - No STP
  - Only one uplink
- **Allow traffic between macvlans (via macvlan stack)**



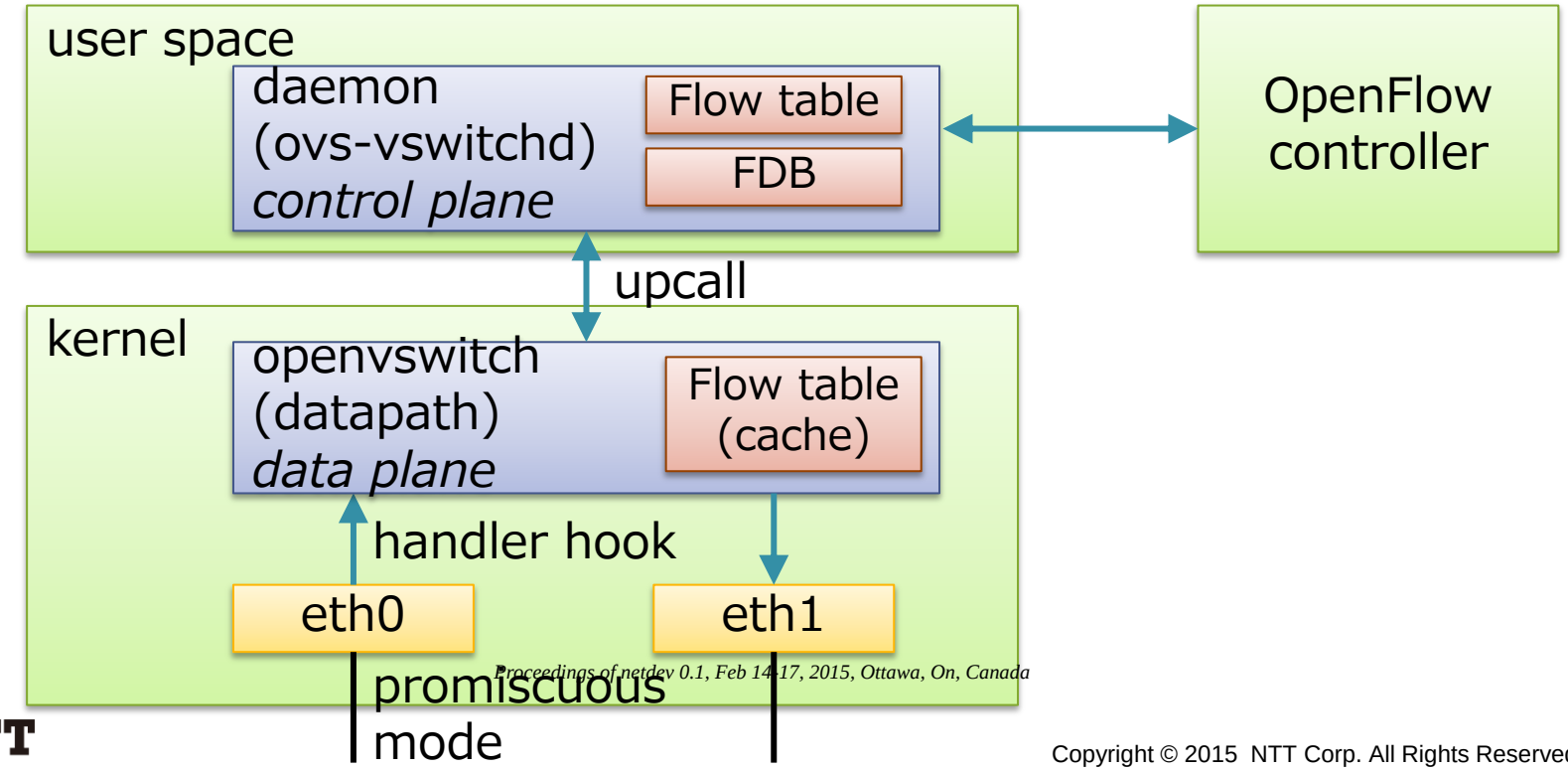
macvtap (private, vepa, bridge) with KVM

- **macvtap**
  - tap-like macvlan variant
  - packet reception
    -> file read
  - file write
    -> packet transmission



Open vSwitch

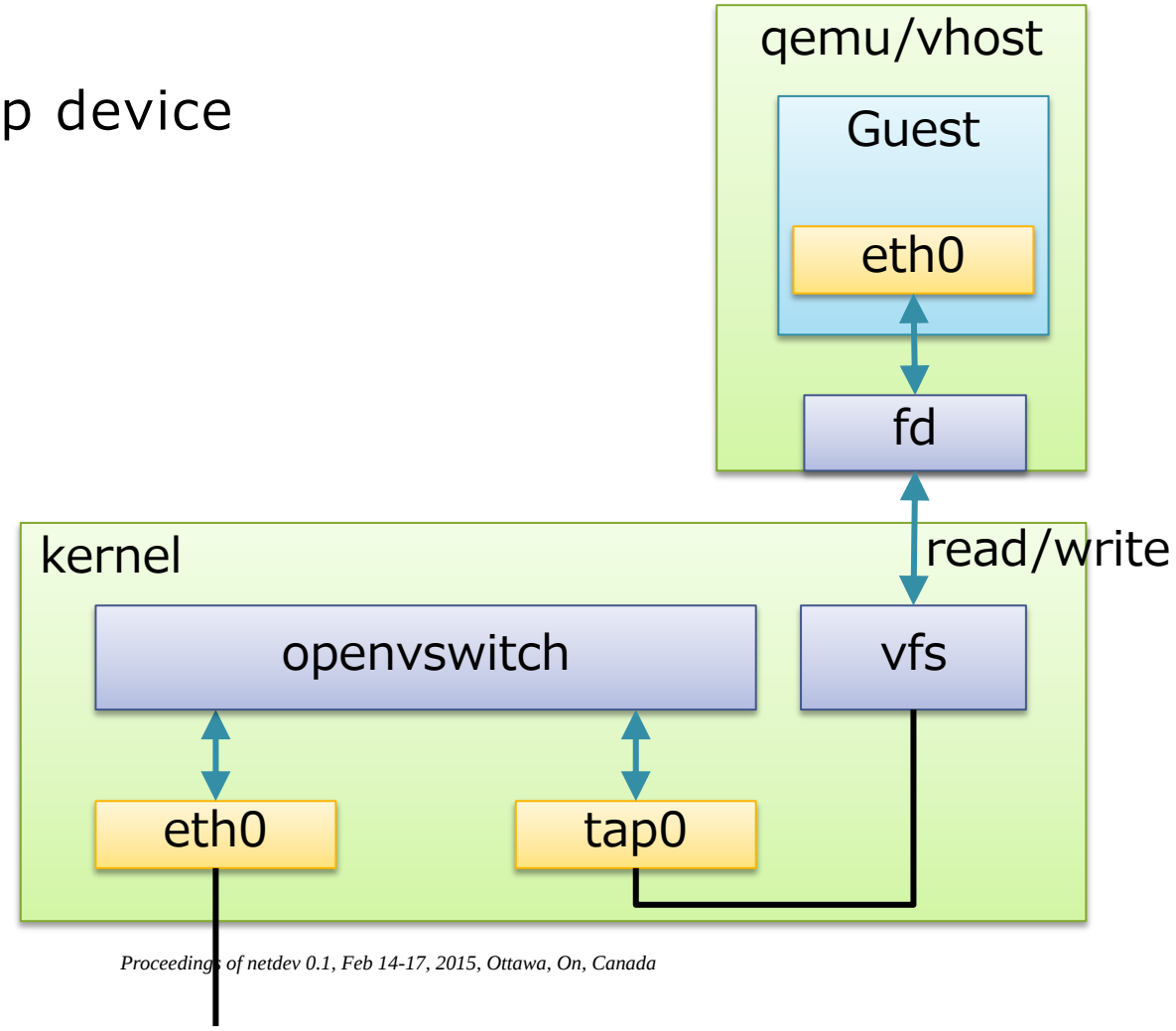
- **Supports OpenFlow**
- **Can be used as a normal switch as well**
  - Has many features (VLAN tagging, VXLAN, Geneve, GRE, bonding, etc.)
- **Flow based forwarding**
- **Control plane in user space**
  - flow miss-hit causes upcall to userspace daemon



Proceedings of netdev 0.1, Feb 14-17, 2015, Ottawa, On, Canada

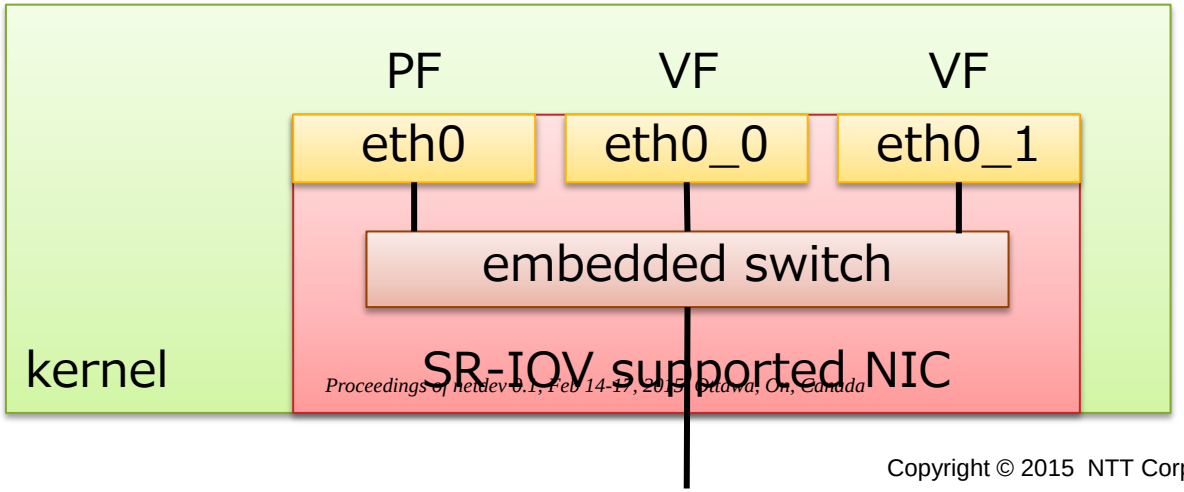
Open vSwitch with KVM

- **Configuration is the same as bridge**
  - used with tap device



NIC embedded switch (SR-IOV)

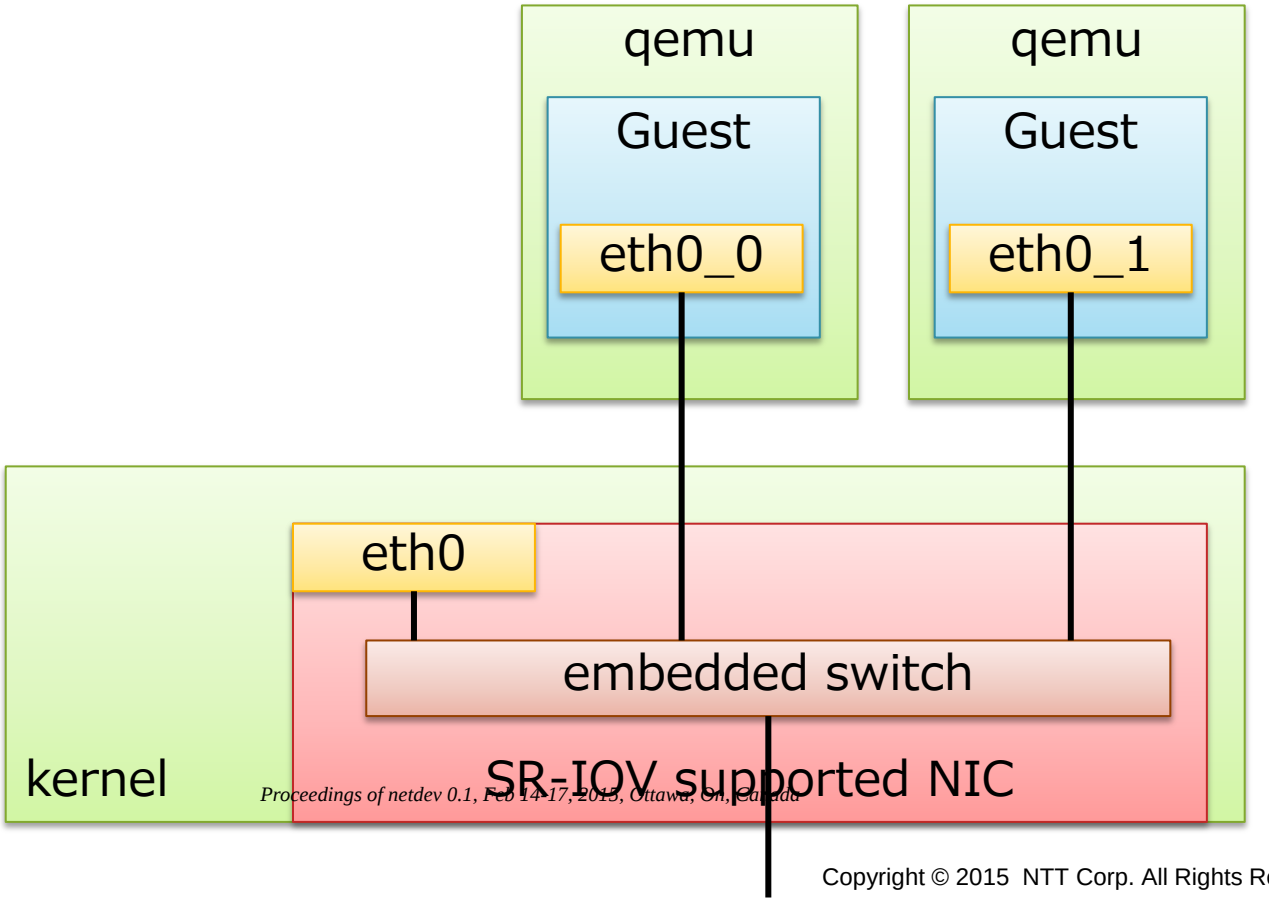
- **SR-IOV**
  - Addition to PCI normal physical function (PF), allow to add light weight virtual functions (VF)
  - VF appears as a network interface (eth0_0, eth0_1...)
  - Some SR-IOV devices have switches in them
    - allow PF-VF / VF-VF communication



Proceedings of netdev 0.1, Feb 14-17, 2015, Ottawa, On, Canada

NIC embedded switch (SR-IOV)

- **SR-IOV with KVM**
  - Use PCI-passthrough to attach VF to guest



Proceedings of netdev 0.1, Feb 14-17, 2015, Ottawa, ON, Canada

Userland APIs and commands (bridge)

Innovative R&D by NTT

- **Various APIs**
  - ioctl
  - sysfs
  - netlink
- **Netlink is preferred for new features**
  - Because it is extensible
  - sysfs is sometimes used
- **Commands**
  - brctl (in bridge-utils, using ioctl / sysfs)
  - ip / bridge (in iproute2, using netlink)

NTT

Userland APIs and commands (bridge)

Innovative R&D by NTT

- **brctl**

```
# brctl addbr <bridge>            ... create new bridge
# brctl addif <bridge> <port>     ... attach port to bridge
# brctl showmacs <bridge>         ... show fdb entries
```

- **These operations can be performed by netlink based commands as well (Since kernel 3.0)**

```
# ip link add <bridge> type bridge   ... create new bridge
# ip link set <port> master <bridge> ... attach port
# bridge fdb show                    ... show fdb entries
```

- **And recent features can only be used by netlink based ones or direct sysfs write**

```
# bridge fdb add
# bridge vlan add
etc...
```


NTT

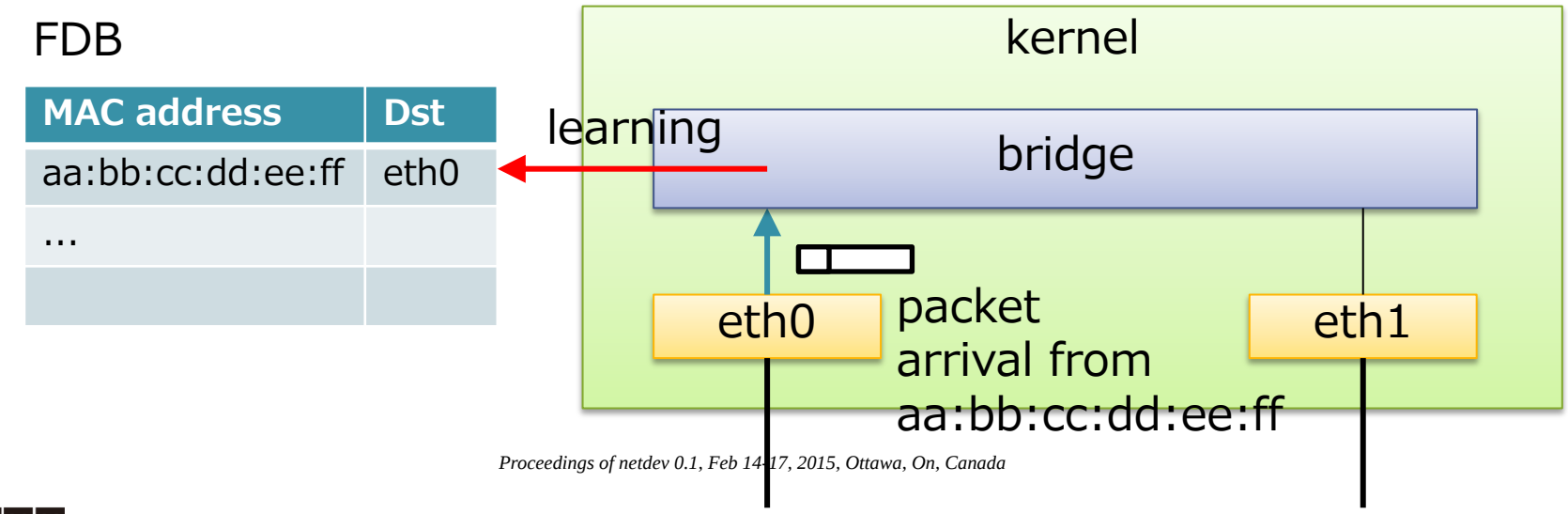
Recent features of bridge (and others)

- FDB manipulation
- VLAN filtering
- Learning / flooding control
- Non-promiscuous bridge
- VLAN filtering for 802.1ad (Q-in-Q)

NTT

FDB manipulation

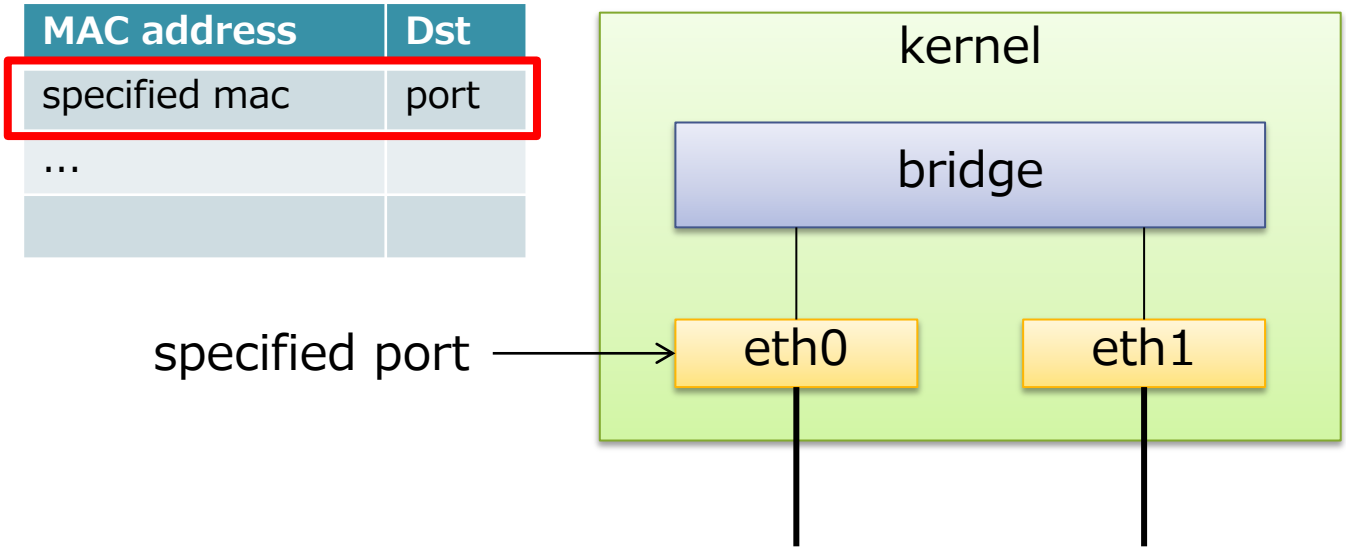
- **FDB**
  - Forwarding database
  - Learning: packet arrival triggers entry creation
    - Source MAC address is used with incoming port
  - Flood if failed to find entry
    - Flood: deliver packet to all ports but incoming one



FDB manipulation

- **FDB manipulation commands**
  - Since kernel 3.0

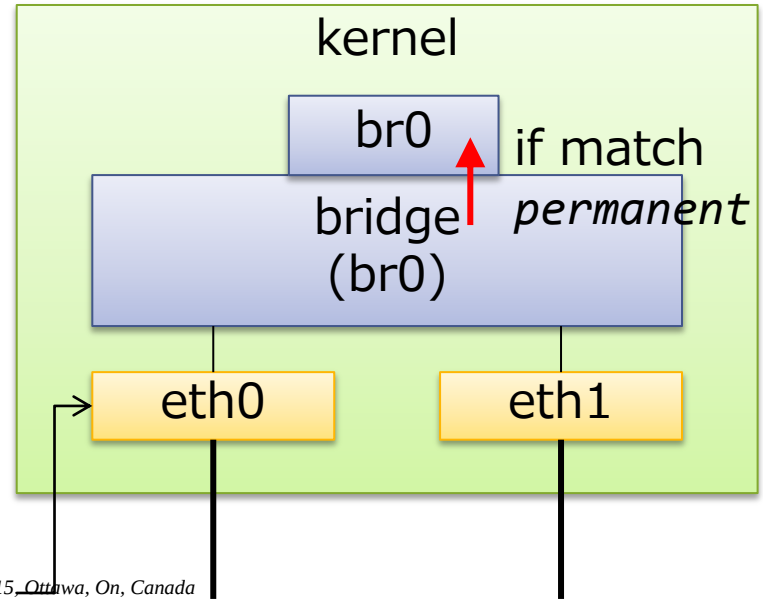
```
# bridge fdb add <mac address> dev <port> master temp
# bridge fdb del <mac address> dev <port> master
```



FDB manipulation

```
# bridge fdb add <mac address> dev <port> master temp
```

- **What's "temp"?**
  - There are 3 types of FDB entries
    - *permanent* (*local*)
    - *static*
    - others (dynamically learned by packet arrival)
  - "temp" means *static* here
  - "bridge fdb"'s default is *permanent*
  - *permanent* here means "deliver to bridge device" (e.g. br0)
  - *permanent* doesn't deliver to specified port



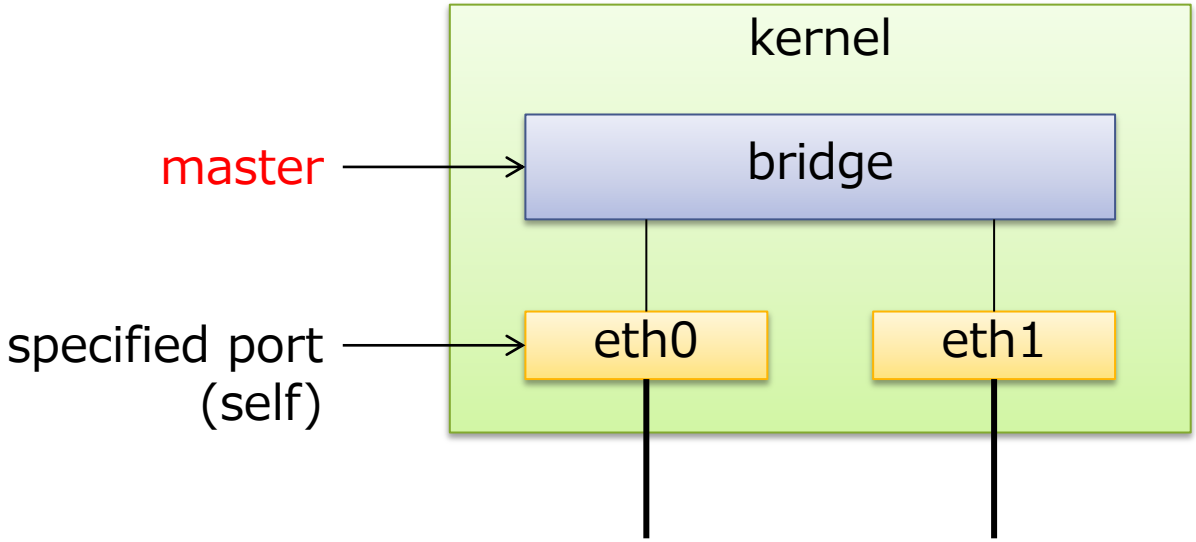
FDB manipulation

```
# bridge fdb add <mac address> dev <port> master temp
```

- **What's "master"?**
  - Remember this command?

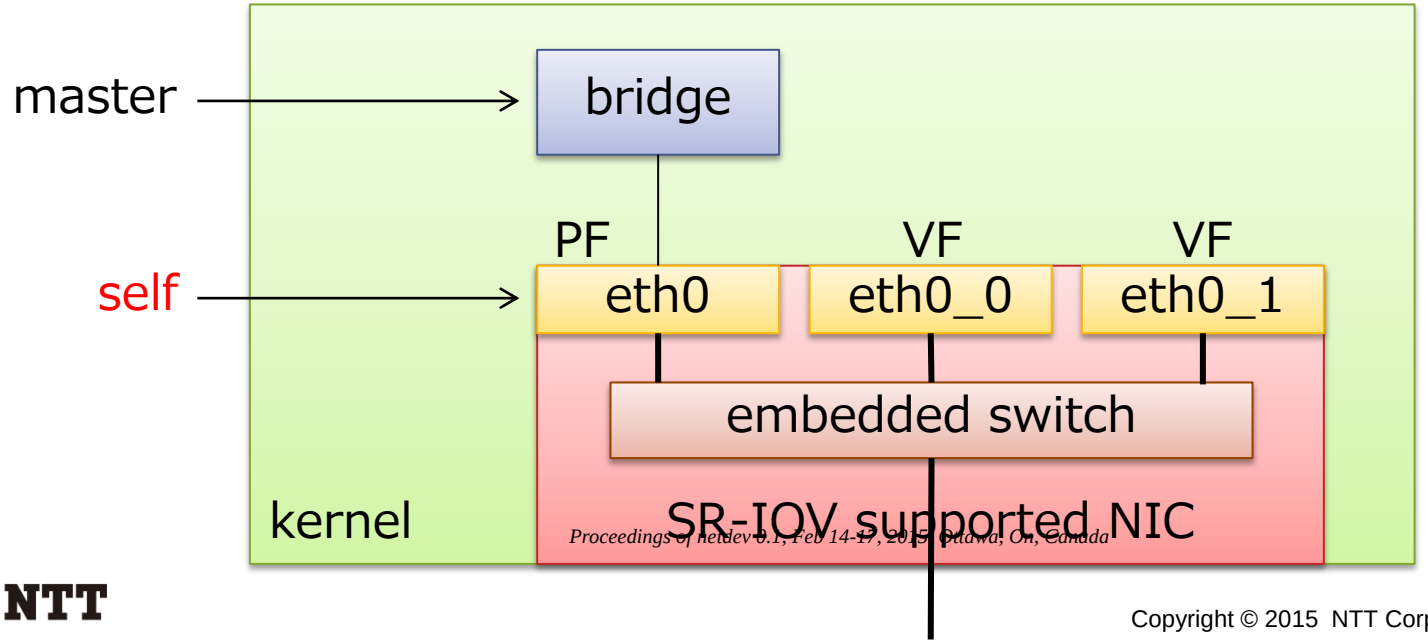
```
# ip link set <port> master <bridge> ... attach port
```

  - "bridge fdb"'s default is "self"
    - It adds entry to specified port (eth0) itself!



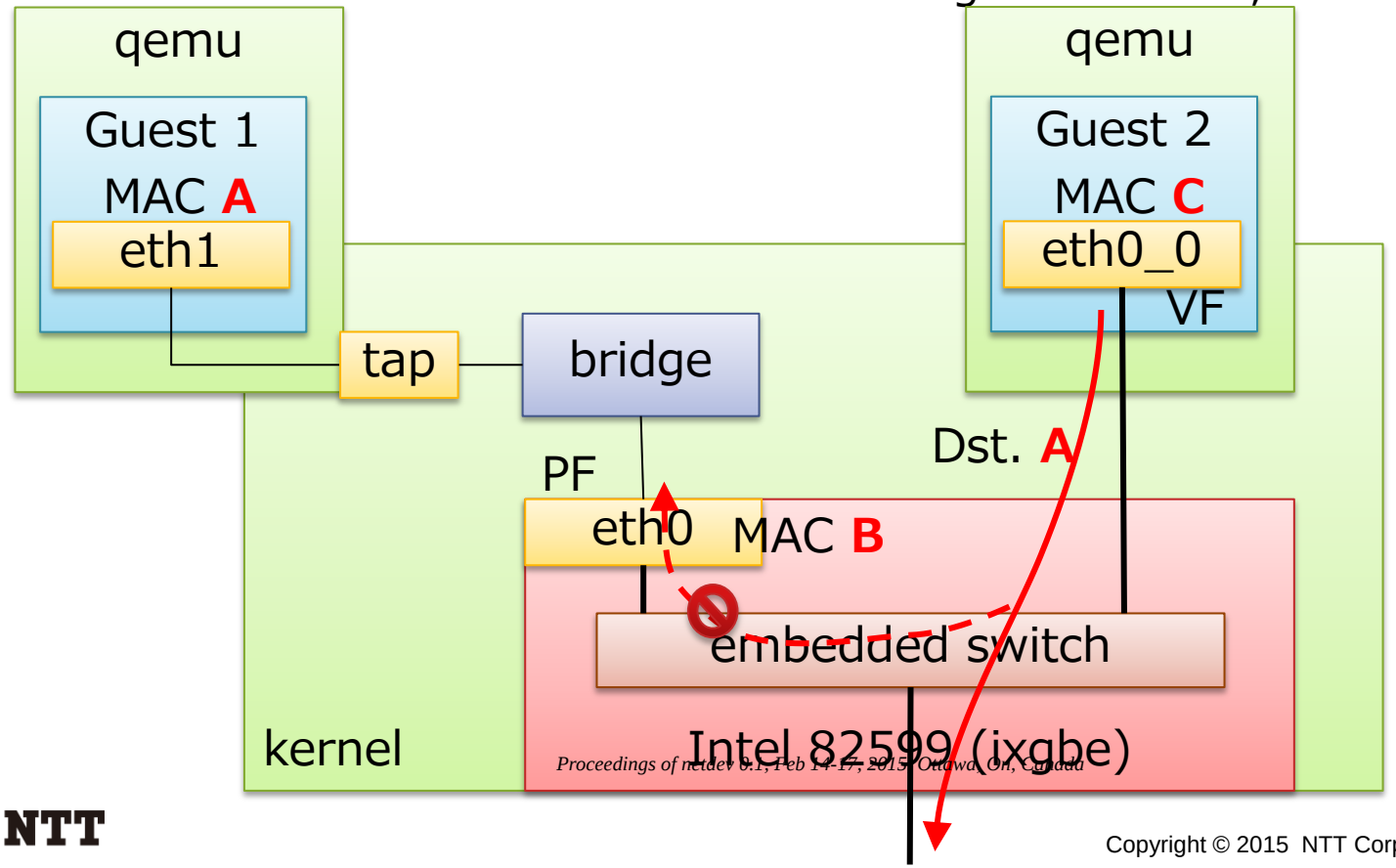
FDB manipulation

- **When to use "`self`"?**
  - Unicast/multicast filtering
    - Use case: SR-IOV embedded SW
  - VTEP-Mac mapping table (vxlan)



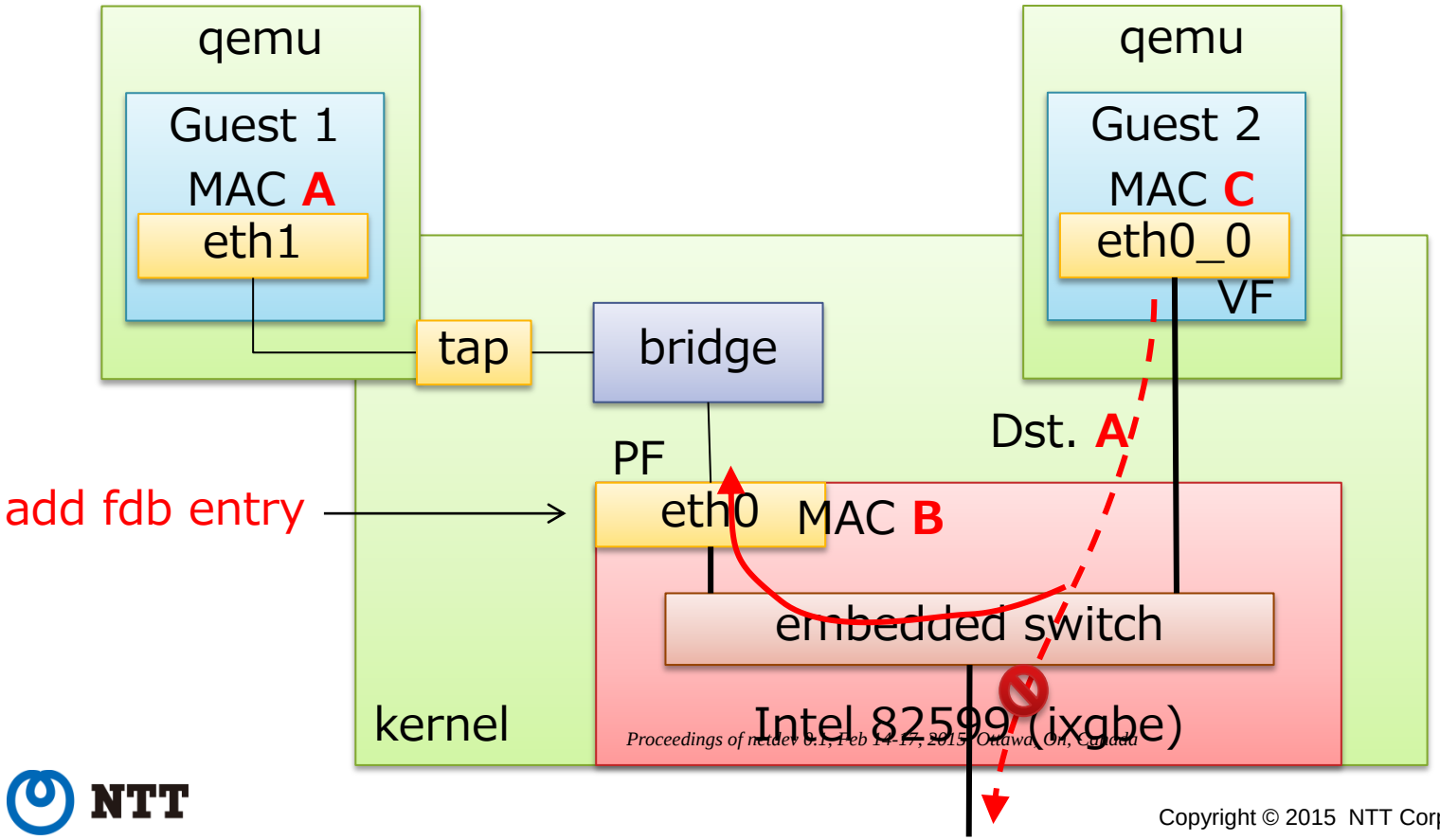
FDB manipulation

- **Example: Intel 82599 (ixgbe)**
  - Some people think of using both bridge and SR-IOV due to limitation of VFs
  - bridge puts eth0 (PF) into promiscuous, but...
    - Unknown MAC address **from VF** goes to wire, not to PF



FDB manipulation

- **Example: Intel 82599 (ixgbe)**
  - Type "`bridge fdb add A dev eth0`" on host
  - Traffic to A will be forwarded to bridge

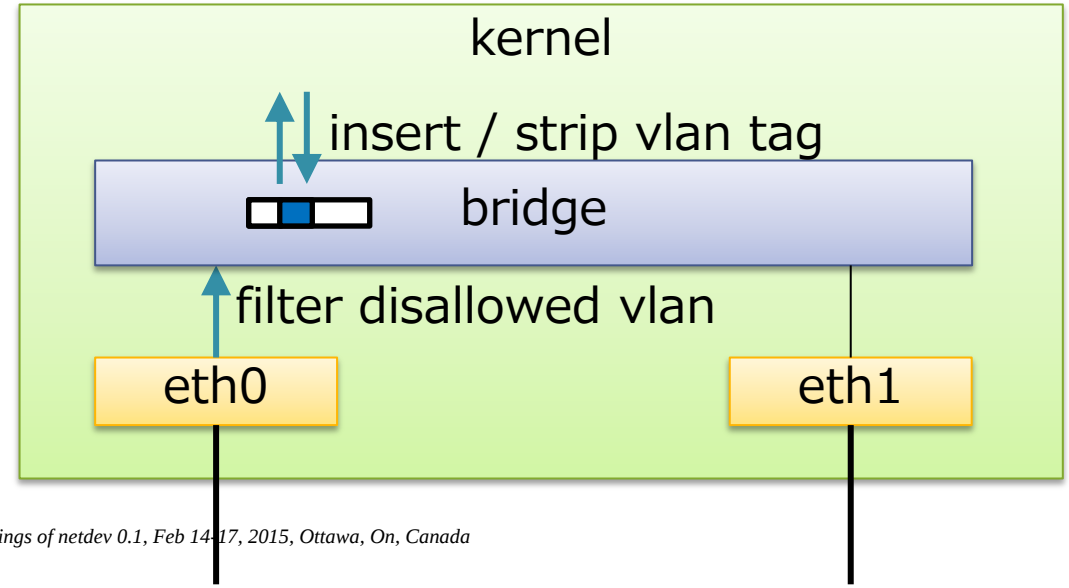


VLAN filtering

- **802.1Q Bridge**
  - Since kernel 3.9
  - Filter packets according to vlan tag
  - Forward packets according to vlan tag as well as mac address
  - Insert / strip vlan tag

FDB

MAC address	Vlan	Dst
aa:bb:cc:dd:ee:ff	10	eth0
…		

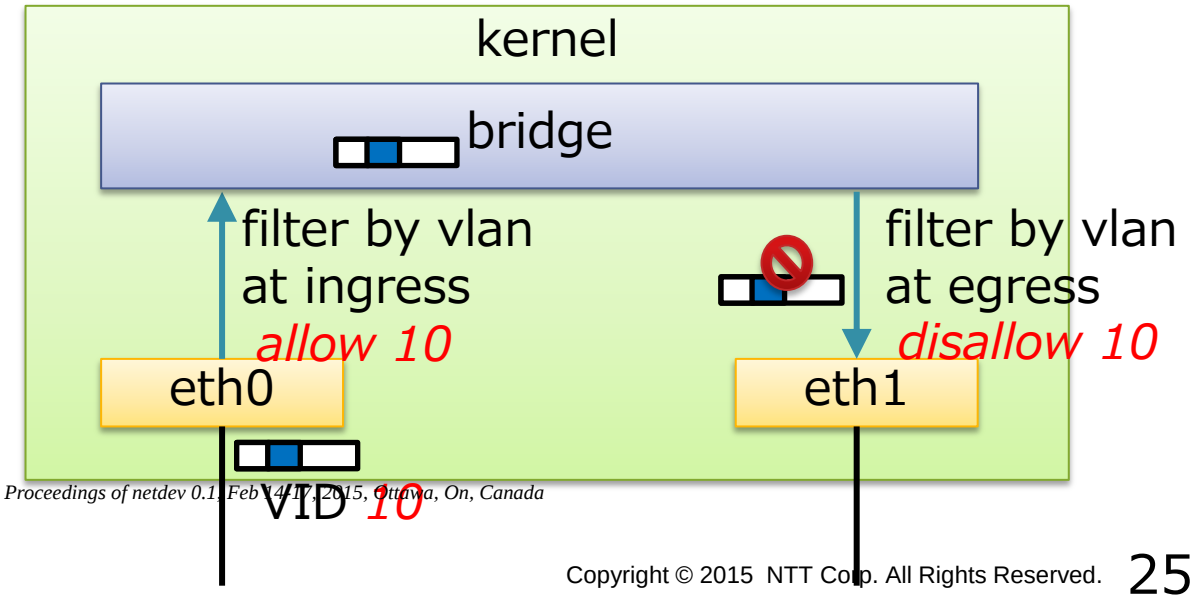


VLAN filtering

- **Ingress / egress filtering policy**
  - Incoming / outgoing packet is filtered if matching filtering policy
  - Per-port per-vlan policy
  - Default is "disallow all vlans"
  - Since kernel 3.18, vid 1 is allowed by default
    - All packets are dropped except for untagged or vid 1

Filtering table

Port	Allowed Vlans
eth0	10
	20
eth1	20
	30

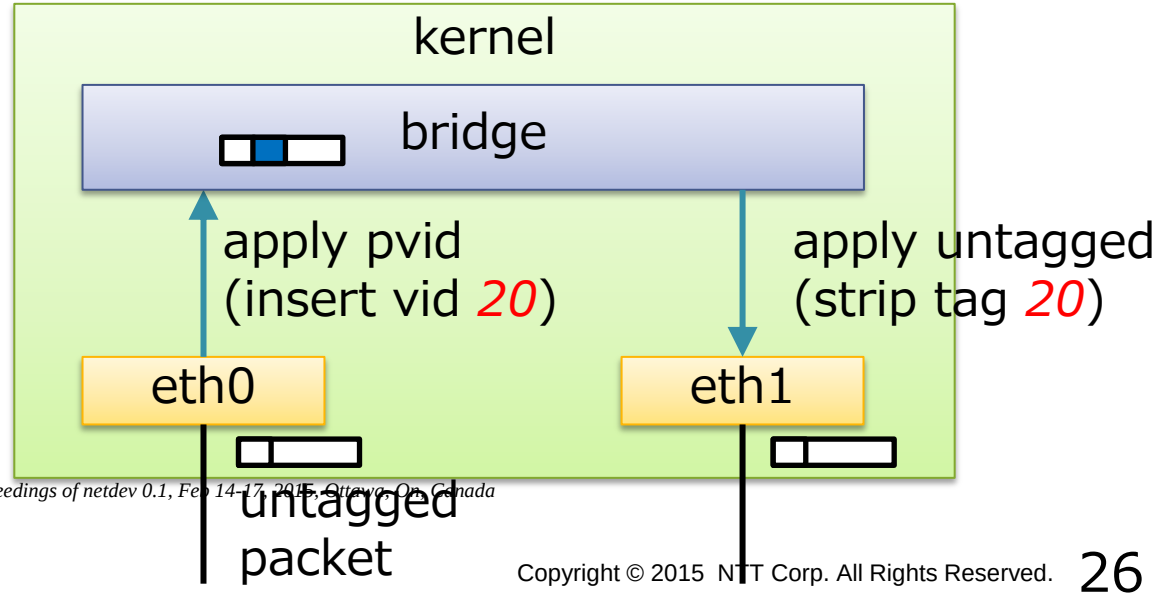


VLAN filtering

- **PVID (Port VID)**
  - Untagged (and VID 0) packet is assigned this VID
  - Per-port configuration
  - Default PVID is 1 (Since kernel 3.18)
- **Egress policy untagged**
  - Outgoing packet that matches this policy get untagged
  - Per-port per-vlan policy

Filtering table

Port	Allowed Vlans	PVID	Egress Untag
eth0	10		✓
	20	✓	✓
eth1	20	✓	✓
	30		



VLAN filtering

Innovative R&D by NTT

- **Commands**
  - Enable VLAN filtering (disabled by default)

```
# echo 1 > /sys/class/net/<bridge>/bridge/vlan_filtering
```

  - Add / delete allowed vlan

```
# bridge vlan add vid <vid> dev <port>
# bridge vlan del vid <vid> dev <port>
```

  - Set pvid / untagged

```
# bridge vlan add vid <vid> dev <port> [pvid] [untagged]
```

  - Dump settings

```
# bridge vlan show
```

- **Note: bridge device needs "self"**

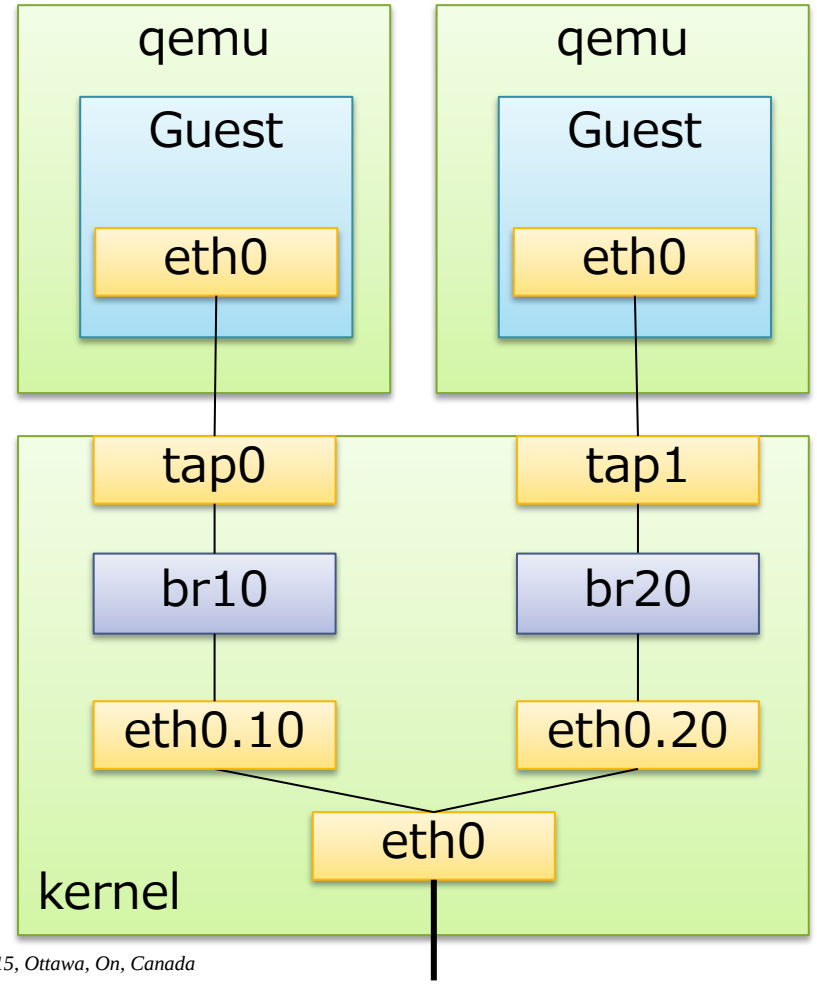
```
# bridge vlan add vid <vid> dev br0 self
# bridge vlan del vid <vid> dev br0 self
```


NTT

VLAN with KVM

- **Traditional configuration**
  - Use vlan devices
  - Needs bridges per vlan
  - Low flexibility
  - How many devices?

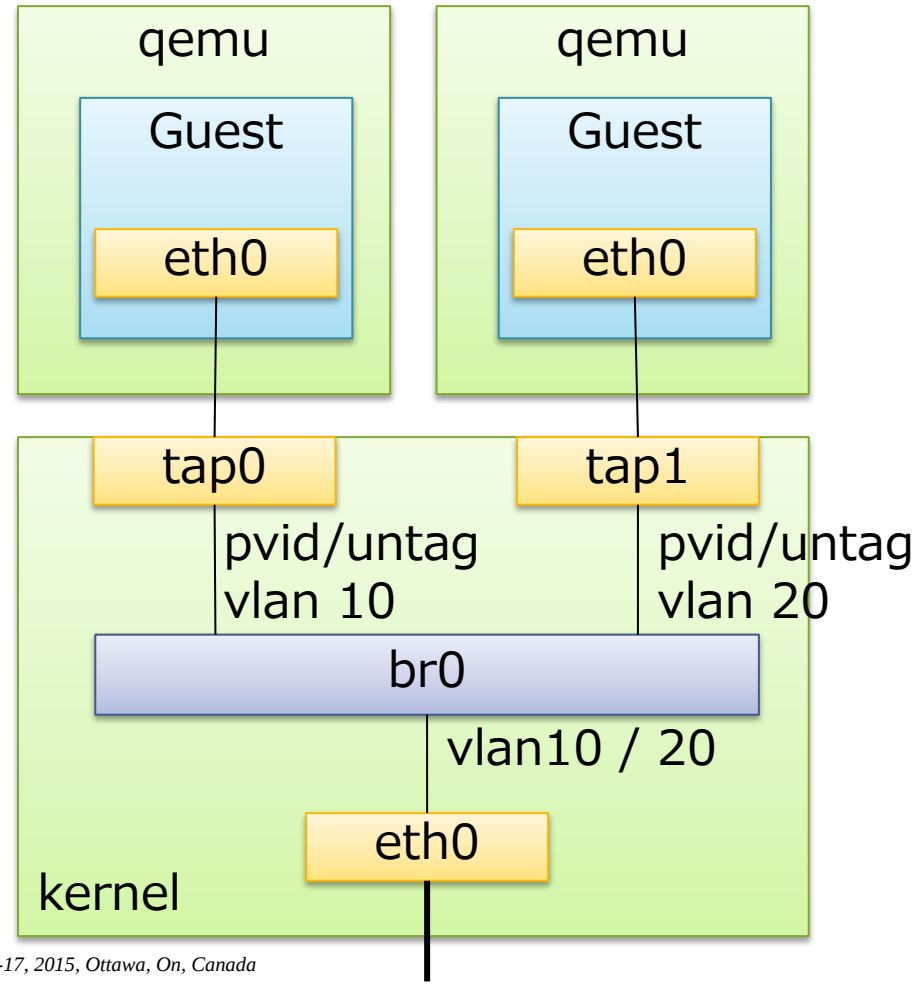
```
# ifconfig -s
Iface ...
eth0
eth0.10
br10
eth0.20
br20
eth0.30
br30
eth0.40
br40
...
```



VLAN with KVM

- **With VLAN filtering**
  - Simple
  - Flexible
  - Only one bridge

```
# ifconfig -s
Iface ...
eth0
br0
```



VLAN with KVM

Innovative R&D by NTT

- **Other switches**
  - Open vSwitch
    - Can also handle VLANs

```
# ovs-vsctl set Port <port> tag=<vid>
```

  - NIC embedded switch
    - Some of them support VLAN (e.g. Intel 82599)

```
# ip link set <PF> vf <VF_num> vlan <vid>
```

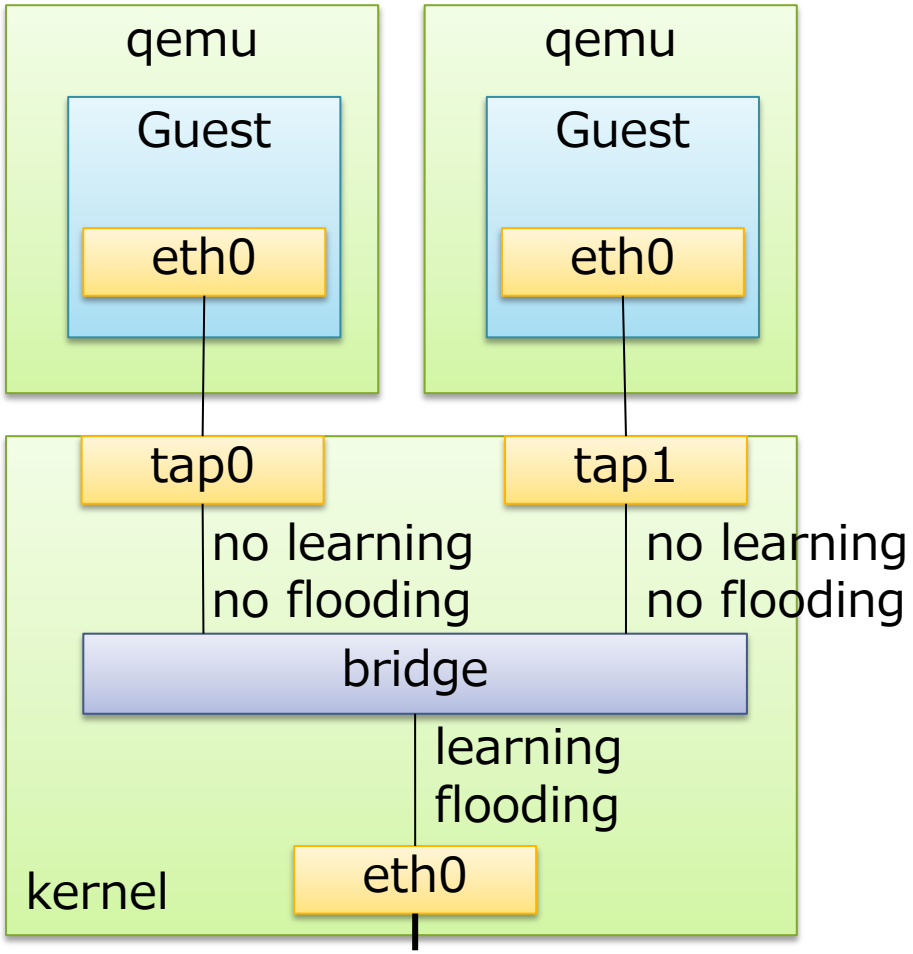

NTT

Learning / flooding control

- **Limit mac addresses guest can use**
- **Reduce FDB size**
- **Used with static FDB entries ("`bridge fdb`" command)**

- **Disable FDB learning on particular port**
  - Since kernel 3.11
  - No dynamic FDB entry

- **Don't flood unknown mac to specified port**
  - Since kernel 3.11
  - Control packet delivery to guests

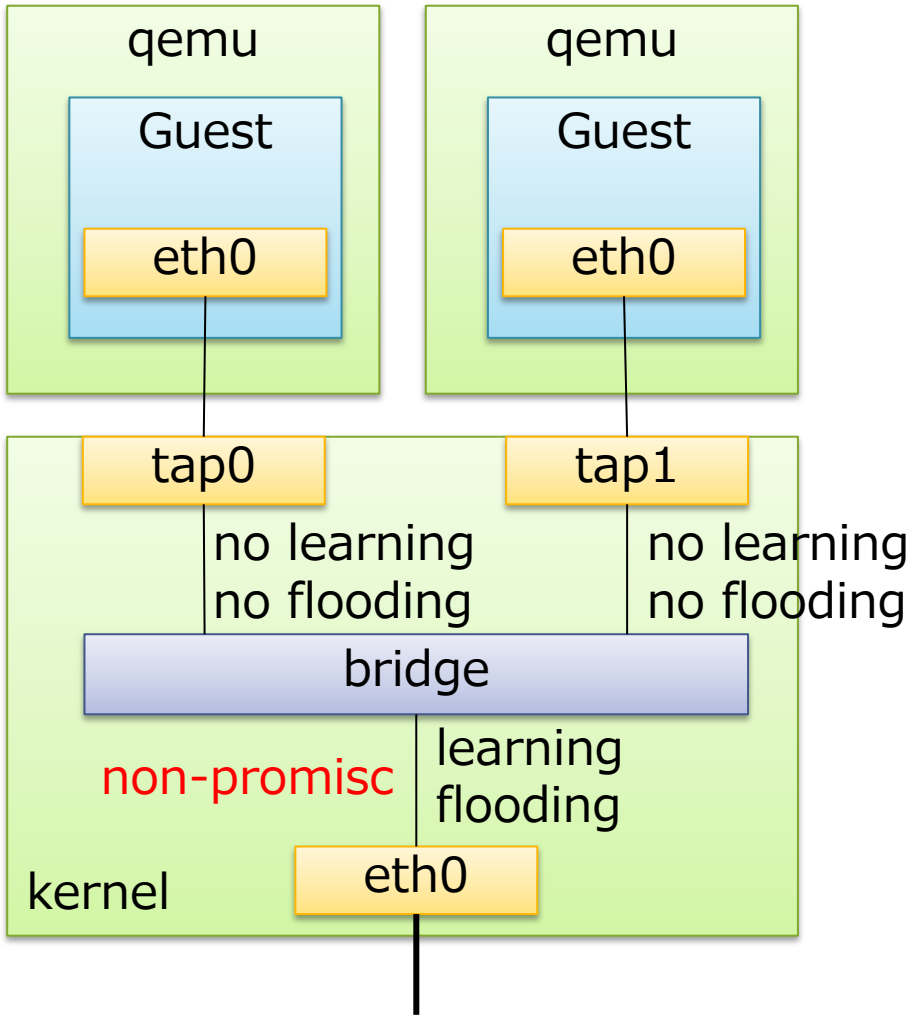


- **Commands**

```
# bridge link set dev <port> learning off
# bridge link set dev <port> flood off
```

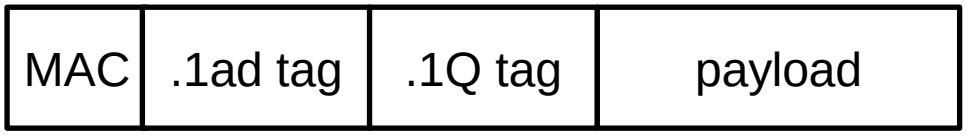
Non-promiscuous bridge

- **Since kernel 3.16**
- **If there is only one learning/flooding port, it can be non-promisc**
- **Instead of promisc mode, unicast filtering is set for static FDB entries**
- **Automatically enabled if meeting some conditions**
  - There is one or zero learning or flooding port
  - bridge itself is not promiscuous mode
  - VLAN filtering is enabled



802.1ad (Q-in-Q) support for bridge

- **Since kernel 3.16**
- **802.1ad allows stacked vlan tags**



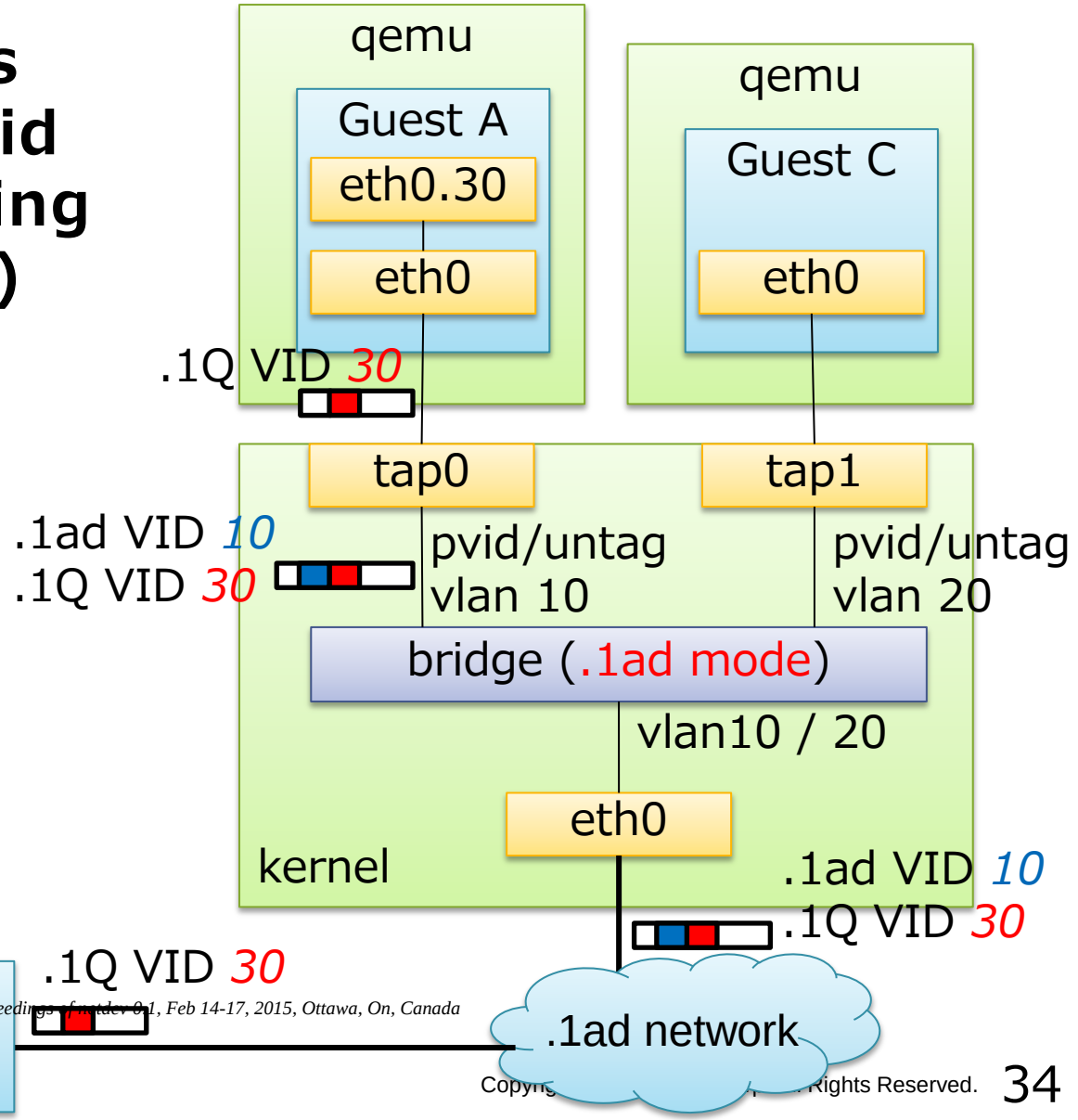
- **Outer 802.1ad tag can be used to separate customers**
  - Example: Guest A, B -> Customer X
    Guest C, D -> Customer Y
- **Inner 802.1Q tag can be used inside customers**
  - Customer X and Y can use any 802.1Q tags
- **Command**

```
# echo 0x88a8 > /sys/class/net/<bridge>/bridge/vlan_protocol
```


NTT

802.1ad (Q-in-Q) support for bridge

- **Bridge preserves guest .1Q tag (vid 30) when inserting .1ad tag (vid 10)**
- **.1ad tag will be stripped at another end point of .1ad network**

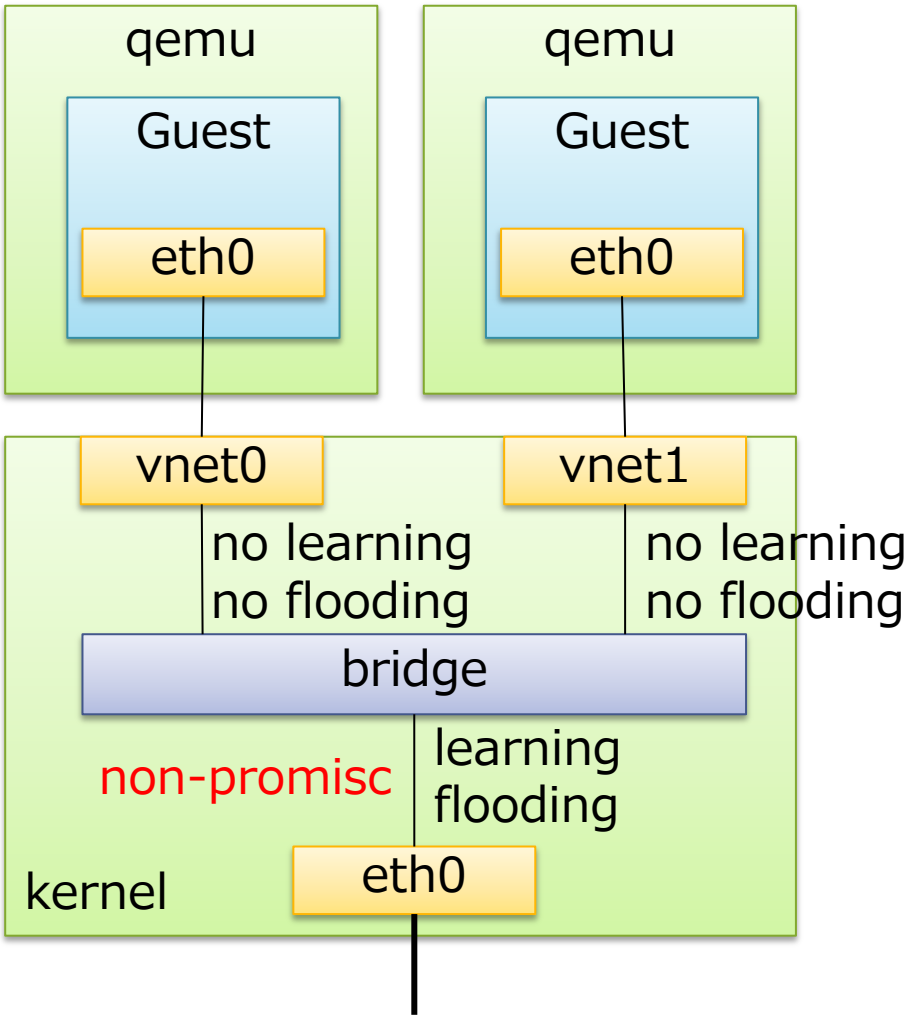


Demo

Non-promiscuous bridge

- **Let's setup non-promiscuous KVM environment!**

- **Steps**
  - Create bridge
  - Enable vlan filtering
  - Attach guests (by libvirt)
  - Add FDB entries
  - Set port attributes (learning/flooding)



Non-promiscuous bridge setup

- **Commands**
  - Create bridge

```
# ip link add br0 up type bridge
# ip link set eth0 master br0
```

  - Enable vlan filtering

```
# echo 1 > /sys/class/net/br0/bridge/vlan_filtering
```

  - Attach guests

```
# virsh start guest1
# virsh start guest2
```

  - Add FDB entries ("append" overwrites if exists)

```
# bridge fdb append 52:54:00:xx:xx:xx dev vnet0 master temp
# bridge fdb append 52:54:00:yy:yy:yy dev vnet1 master temp
```

  - Set port attributes

```
# bridge link set dev vnet0 learning off flood off
# bridge link set dev vnet1 learning off flood off
```

Non-promiscuous bridge via libvirt xml

- **libvirt (>= 1.2.11 with kernel >= 3.17) can automatically handle these settings**
  - Network XML

```
# virsh net-edit <network>
...
  <bridge name="br0" macTableManager="libvirt"/>
...
```

Some more useful commands...

- **Filter FDB dump per bridge/port (Since 3.17)**
  - Filter per bridge

```
# bridge fdb show br <bridge>
```

  - Filter per port

```
# bridge fdb show brport <port>
```

- **VLAN range (Coming soon... 3.20?)**
  - Add vlans

```
# bridge vlan add vid <vid_begin>-<vid_end> dev <port>
```

  - Show vlans in compressed format

```
# bridge -c vlan show
```

Summary

- **Linux has several types of switches**
  - bridge, macvlan (macvtap), Open vSwitch
  - SR-IOV NIC enbedded switch can also be used

- **Bridge's recent features**
  - FDB manipulation
  - VLAN filtering
  - Learning / Flooding control
  - Non-promiscuous bridge
  - 802.1ad (Q-in-Q) support