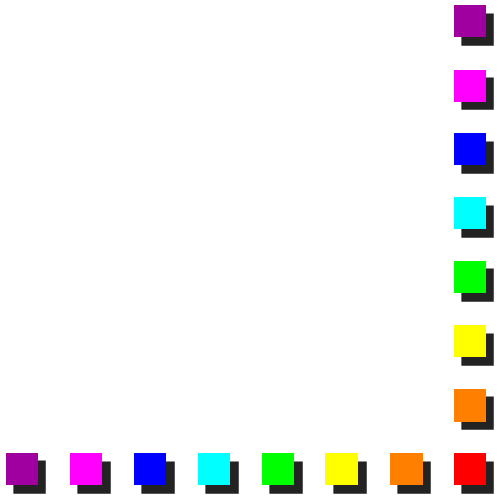
Introduction to Virtual Networking

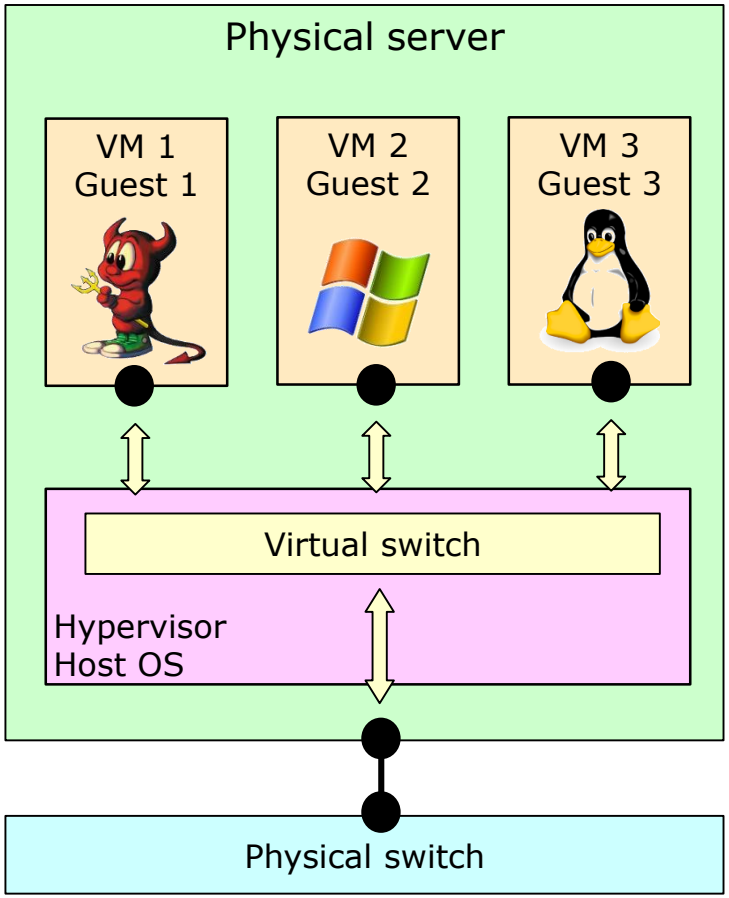
Fulvio Risso

Politecnico di Torino



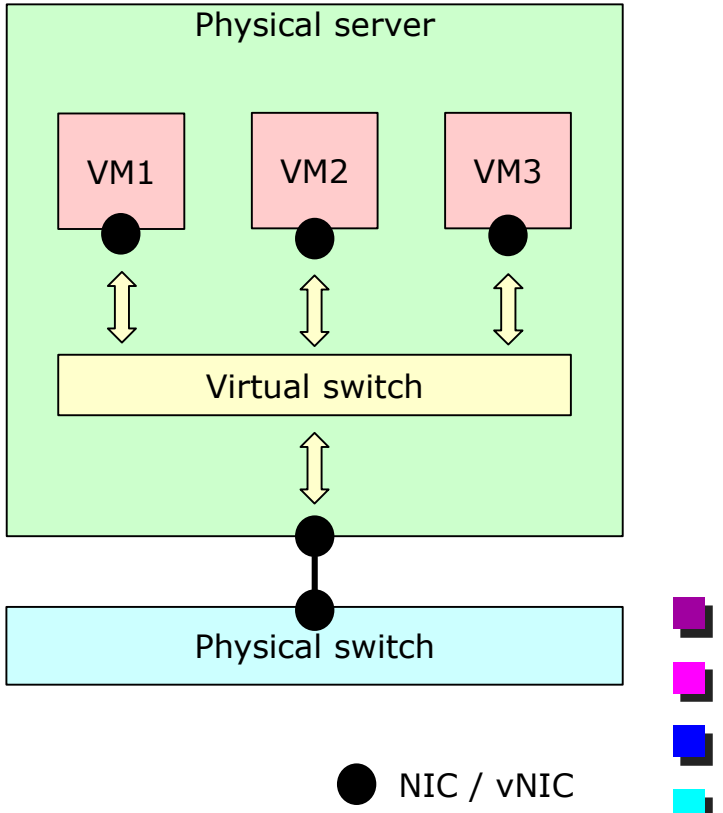
Computing virtualization in brief

- Capability to run multiple virtual machines on the same physical host
- Main software components
  - Virtual machines
  - Host OS / Hypervisor
  - Software (virtual) switch (or software bridge)
- All those components (including the virtual switch) are created by OS people



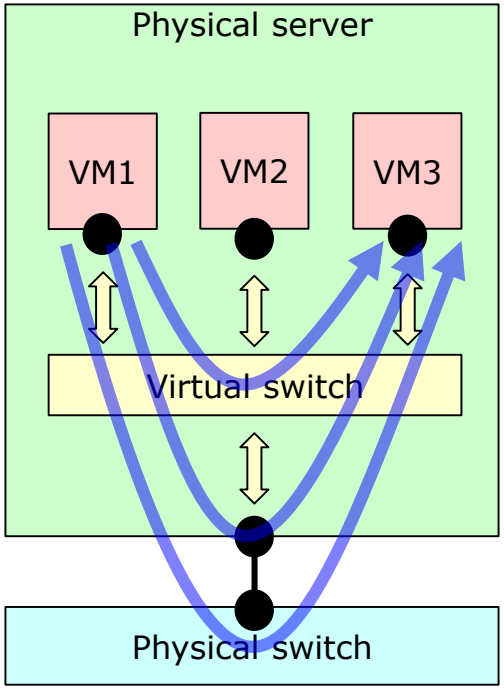
Virtualization and networking

- Virtualization introduces additional complexity for networking as well
- Necessity to:
  - Deliver traffic from VMs to the physical network (and vice versa)
  - Deliver traffic between VMs within the same server



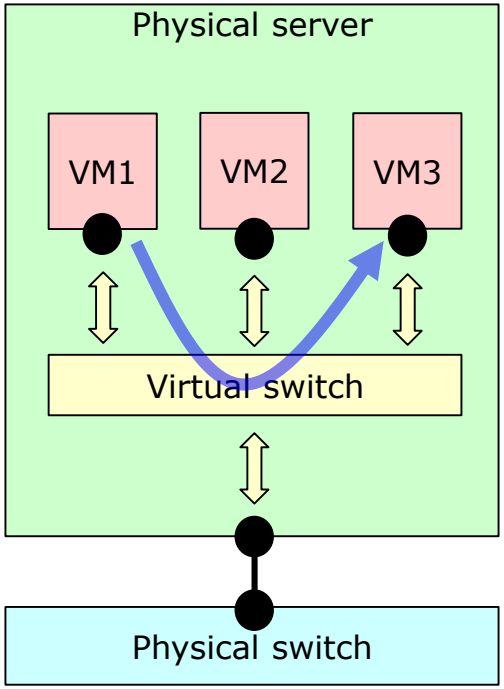
Communication between Virtual Machines

- Three main options
- Virtual switch in the host OS / hypervisor
  - Historically, the first solution used
  - In some cases the software switch is in the Host OS, in some other in the Hypervisor (e.g., in case in case the virtualization layer sits on top of the operating system such as in Virtualbox)
- In the NIC (using virtual queues)
- In the external switch (requires special support)



1. Host-based switching

- Strengths
  - High bandwidth (and reduced overhead) for inter-VM traffic
  - Enforces policies early
- Weaknesses
  - Additional processing overhead (for the switch)
  - Additional switch to configure and monitor
  - Historically, the switch *WAS* feature-weak
  - Not clear who controls the switch: IT or networking people?

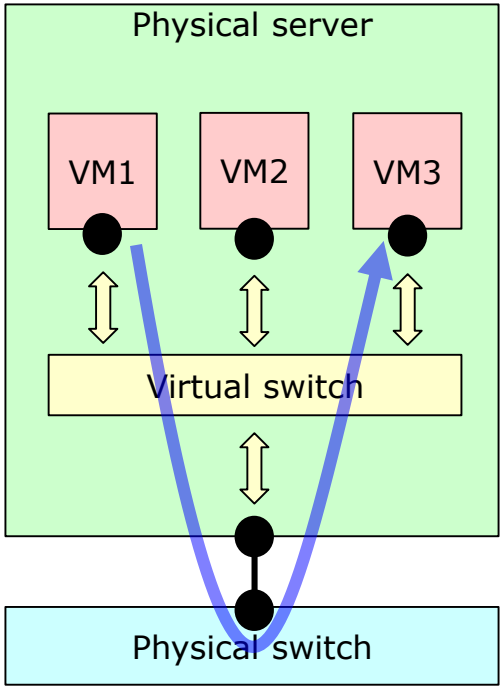


Pros and cons

- Centralized management
- Can implement the same features of hardware switches
  - Visibility
  - ACLs
  - Quality of Service
- Can support a larger number of policy rules
- Consumes hypervisor CPU cycles
- Sometimes, even controlled by the same commands (e.g., Cisco IOS) and through the same management system that controls the rest of the infrastructure
  - vSwitch can be seen as a native piece of the infrastructure
- Possible hardware offloading can provide big wins
  - Checksum and TSO offloading
  - SR-IOV may be even better
- Examples
  - VMware vSwitch, Cisco Nexus 1000V, Open vSwitch

2. Hairpin switching

- Use hardware that is already present
- However, hardware must be compatible with hairpin switching
  - Need to modify the bridge forwarding mechanism: 802.1D bridges never forward a frame on the same port on which it has been received!
- Performance
  - Can leverage the power of the hardware (i.e., forwarding speed)
  - Traffic crosses twice all the intermediate following components
    - Hypervisor, NIC, physical link, …
  - Hence in the end it may not be so convenient, particularly for VM-to-VM traffic
- Apparently it does not consume CPU cycles
  - The traffic has to be delivered to the NIC (and from there to the physical switch), and this has a cost
- "hairpinning": traffic makes a U-turn, looking like a "hairpin"

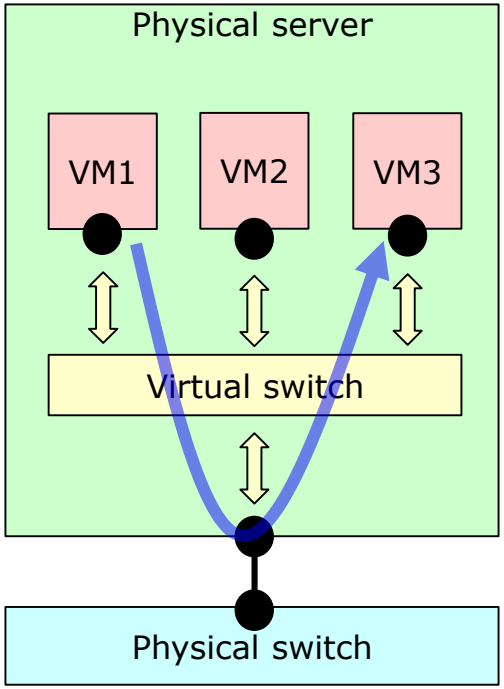


Traffic tagging

- How can the switch distinguish traffic coming from different VMs?
  - Not a problem for software switches: they know exactly which VM generated each frame
  - May be a problem for hardware switches (NIC and hairpin)
  - We need to "tag" the traffic in some way before sending traffic to the network
- Options for hardware switches
  - Based on the MAC address of the vNIC: not very secure, can be easily spoofed
  - Softswitch can add a tag to each frame (e.g., vNIC-based tag)
    - Tag space limited
    - It may cause issues with multicast and mobility
- This problem is the same for both hairpin and NIC switching

NIC switching

- Intermediate solution between host and hairpin switching
  - Intermediate also in terms of strengths and weaknesses
- Several NICs available supporting this feature
- Most important advantage: avoid the problem about *who* controls edge switches
  - NICs are not under the control of the Operating System, hence can be considered part of the network infrastructure

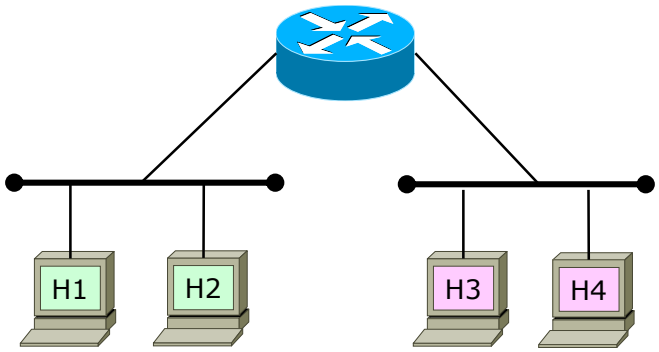


Hardware vs Software switching: performance

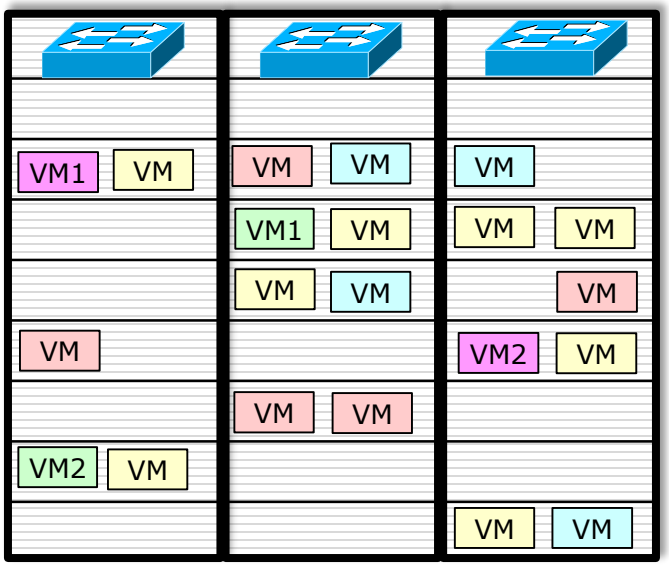
- Software switches have been demonstrated at Nx10Gbps
  - Although with significant CPU overhead, stolen from VM processing
    - But… are we creating this infrastructure to achieve fast switching processing or to support many VMs?
- Software switches can drop traffic closer to the source
  - Important in clouds with over-subscribed links and untrusted sources
- Hardware offloading for software switches can provide big wins
  - Checksum and TSO offloading
  - SR-IOV may be even better
- Software switches are better for local VM-to-VM traffic

Impact of Virtualization on Networking (1)

- With virtualization… everything looks possible
  - Example 1: two VMs of the same tenant running on (distant) servers
  - Example 2: VM migration used to optimize computing tasks, energy consumption, etc…



Physical world: the network follows the physical topology



Virtual world: the network is completely separated from the physical topology

Impact of Virtualization on Networking (2)

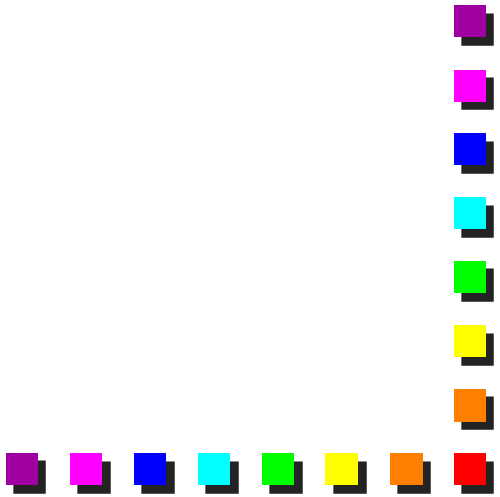
- Necessity to create a **bridged network** spanning across the entire data center in order to support the flexibility enabled by the virtualization
  - L3 introduces segmentation, hence introduces some additional problems
- Possible problems with data-link networks
  - Data-link networks (and VLANs) do not scale with the number of expected VMs (even millions!)
  - VLAN tags are not enough (QinQ?)
  - Policies (either in the softswitch or in the network) do not follow VM migration
  - Need to propagate VM contexts (e.g., tags) across the infrastructure

Conclusions

- Hairpin switches attractive when applying similar policies over all nodes or in aggregate with little local VM-to-VM traffic
  - Clear separation between computing and networking domains
- Host switches provide more flexibility and fine-grained control at cost of hypervisor CPU cycles
  - Not clear separation between computing and networking domains
- Host switches is currently the most common solution
  - Commercial interests are pushing for this solution, adding the most intelligence we can in them, leaving the network to act as a dumb pipe (just provide basic connectivity)
  - Easy to add new features (e.g., added value services such as mobility), without requiring explicit support from the network
    - Overlay model

Section 2

Software bridges in Linux



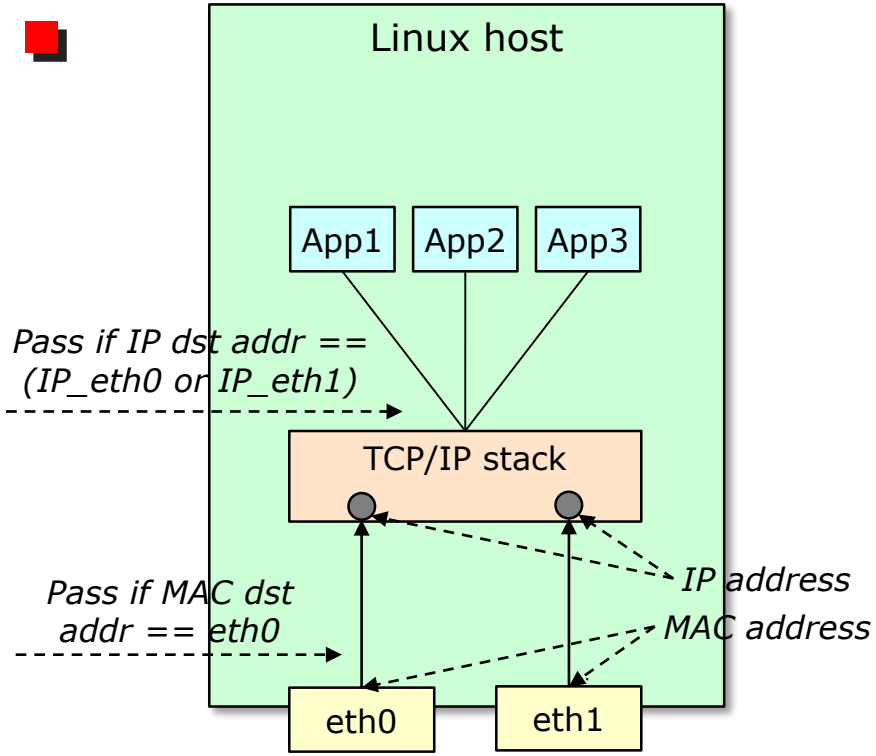
Introduction

- Linux has 3 types of software bridges
  - Linuxbridge
  - macvlan
  - Open vSwitch

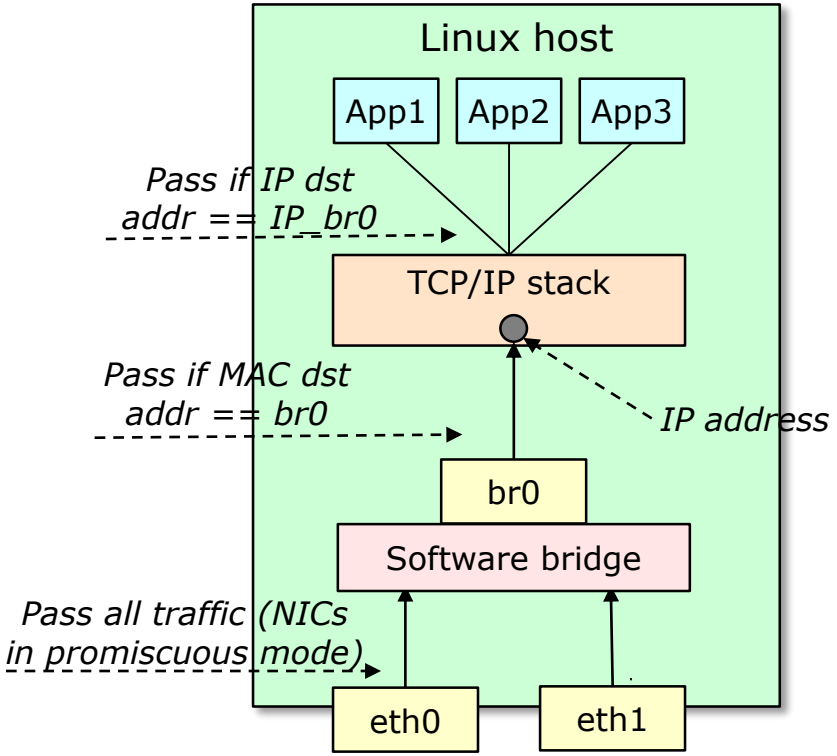
- Why do we need a software switch?
  - 1. Provide intra-host connectivity to execution containers (e.g., different network namespaces, VMs, Docker, Linux Containers, etc)
  - Handle IP addresses differently from the “official” IP model
    - “One IP address per NIC” rule

Basic bridging networking in Linux (1)

- Linux allows to create a bridge across the interfaces, and one unique (virtual) interface that is valid for the entire host



Traditional networking: IP addresses assigned to the NIC



Bridged networking: IP addresses assigned to the bridge

Basic bridging networking in Linux (2)

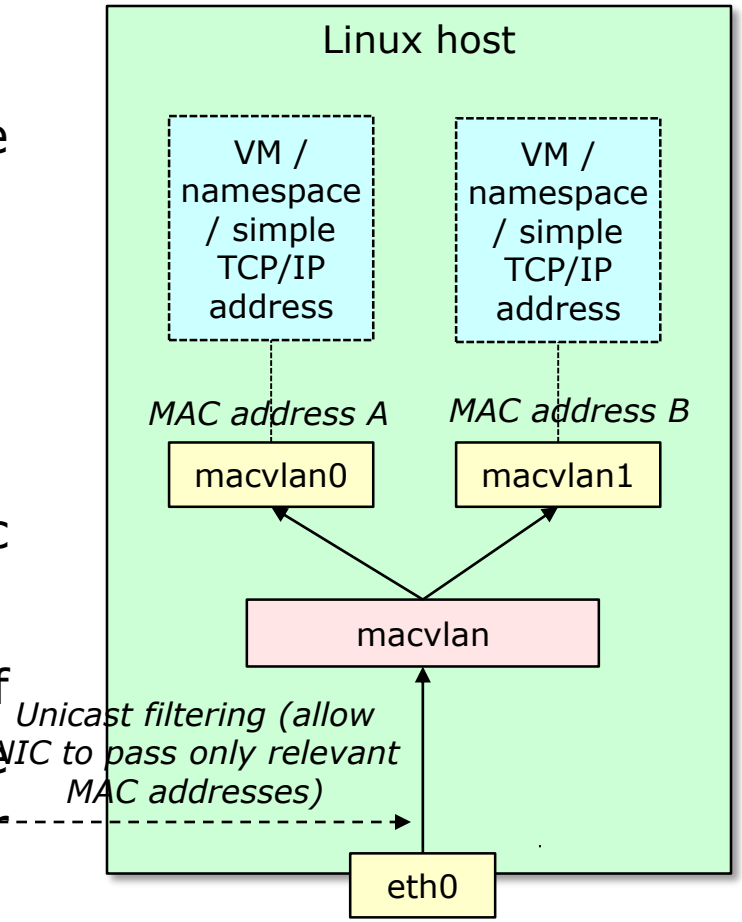
- In the past: the setup of a bridging process in Linux was a way to have an 802.1D bridge, running in software, that was part of a bigger network
  - We can modify the bridge code
  - We can play with a bridged network without having to buy a physical device
- Now: it represents a way to overcome one of the limitations of the IP protocol, which requires a *distinct* IP address on each interface
  - Since the default gateway is unique for the entire host, it is hard to use multiple paths
    - It requires an ICMP redirect or the manual definition of a specific route
  - If allowed by the STP, all the NICs can be used to rx/tx traffic

Linuxbridge

- Most common software bridge: *linuxbridge*
- Behaves like a traditional hardware switch (i.e., IEEE 802.1D)
  - Has filtering DB (FDB), spanning tree (STP), etc.
- NICs in promiscuous mode allow to receive all the packets
  - Filtering based on the MAC address of the *br0* virtual device
- Traffic can be sent to the network using both network interfaces
- The IP address configured on the bridge is reachable from both interfaces
  - Looks like a “loopback” address

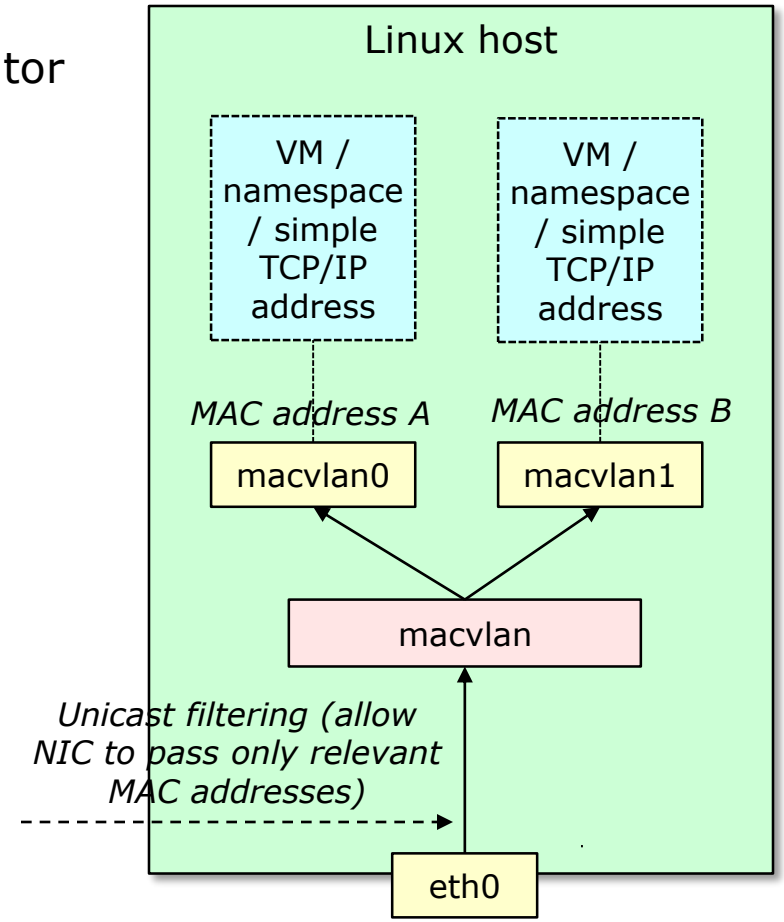
Macvlan (1)

- Implements a VLAN-like behavior by using MAC addresses instead of 802.1Q tags
- Looks like a (L2) subinterface derived from the main NIC
- Each macvlan interface
  - Has its own MAC address
  - Can have a distinct IP address
- Applications can bind to a specific interface / IP address
- LXC guests can use one side of that interface, by moving the macvlan interface in their namespace



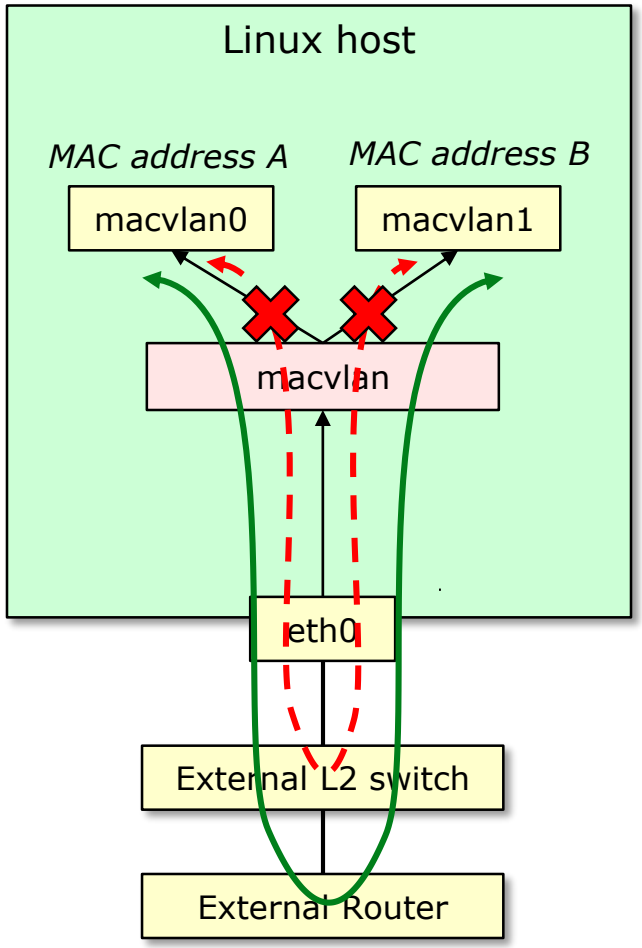
Macvlan (2)

- Supports four operating modes
  - Private
  - Virtual Ethernet Port Aggregator (VEPA) - Default mode
  - Bridge
  - Pass-through
- Can configure the NIC to filter unicast packets based on MAC address instead of running in promiscuous mode (except for pass-through)
  - Requires explicit NIC support



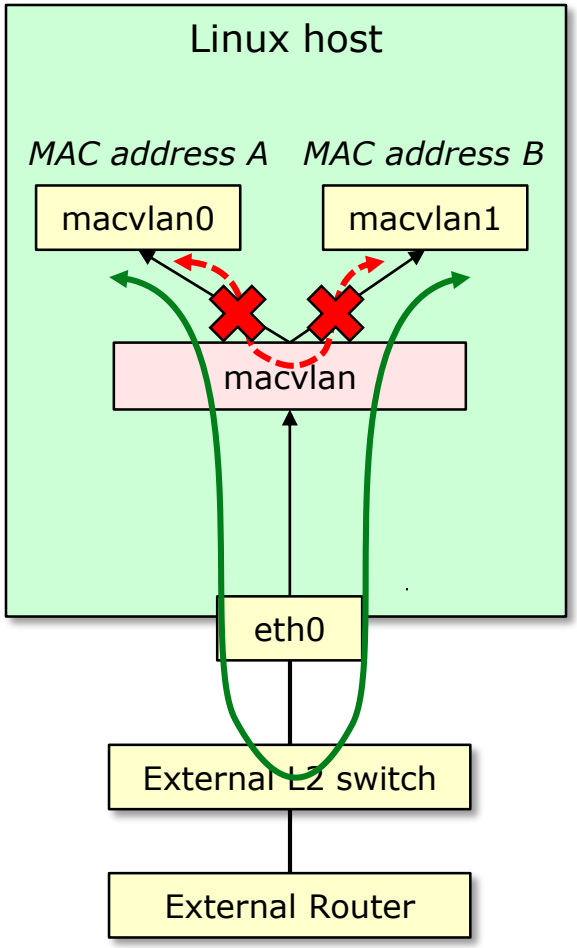
Macvlan: private mode

- VLAN-like behavior
- Forbids any inter-macvlan traffic
  - I.e., MAC A cannot send data to MAC B
- Traffic will be delivered only if it comes with a different source MAC address
  - E.g., from MAC A to a router and then to MAC B



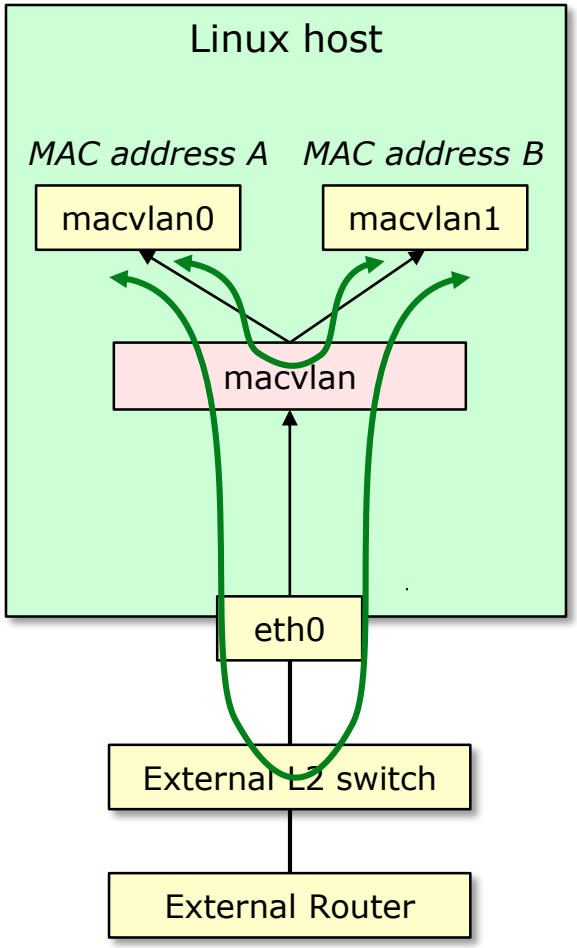
Macvlan: VEPA mode

- In this case MAC A is allowed to talk with MAC B only if the traffic comes from the external world
  - I.e., the Macvlan driver cannot send traffic from MAC A to MAC B
  - However, if the same frame is received from an external switch (it may also be the L2 switch on the NIC), the packet is delivered correctly
- The problem is that the external switch cannot be a traditional 802.1D bridge
  - 802.1D does not allow a frame to be sent back to the same interface on which it was received



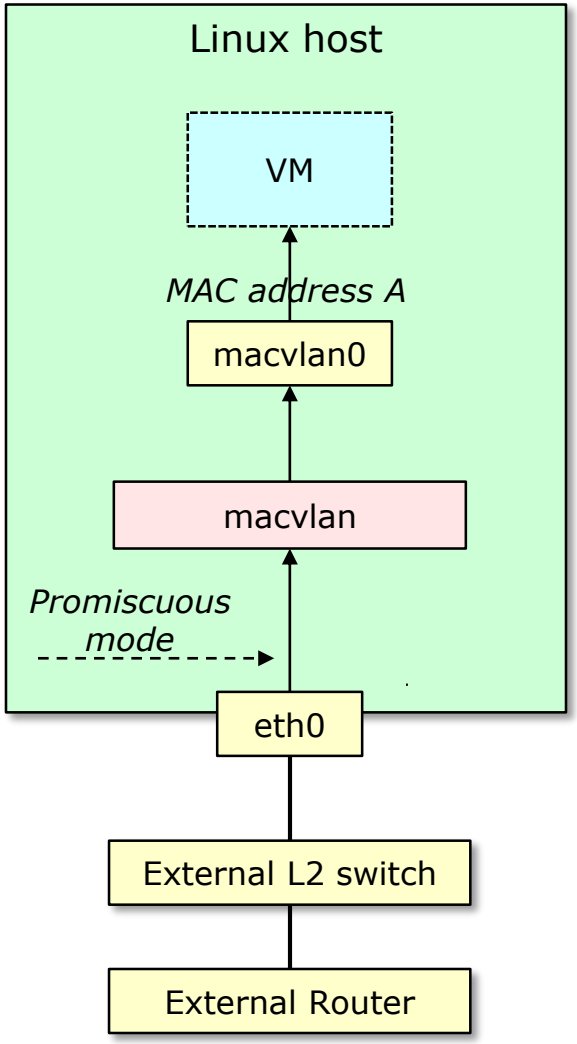
Macvlan: bridge mode

- Allows forwarding of all the traffic
- The Macvlan driver acts as a (simple) bridge
  - No filtering database (i.e., MAC learning)
  - No STP
  - Only one uplink



Macvlan: pass-through mode

- Supports only one macvlan device
- Allow all traffic to be sent to the upstream component (e.g., VM)
  - Used mainly for VMs (as macvtap)
- NIC has to work in promiscuous mode
- Allow the upstream component to use any MAC address
  - No MAC filtering in the macvlan driver, nor in the NIC
  - Traffic is always sent upstream, and that component (e.g., the VM vNIC driver) will be in charge of the filtering
- MAC A is no longer used



OpenvSwitch

- Visibility (NetFlow, sFlow, SPAN/RSPAN)
- Fine-grained ACLs and QoS policies
- Support for both traditional L2 forwarding and OpenFlow
- Port bonding, GRE, and IPsec
- Works on Linux-based hypervisors: Xen, XenServer, KVM, VirtualBox
- “Native” in Linux
- Open source, commercial-friendly Apache 2 license
- Multiple ports to physical switches