

Hardware vs. Software Routers

Oliver Knapp, Consulting Engineering Nokia

DENOG9 - 24.11.2017

Agenda

1. Why software routers at all ?
2. Technology: hardware routers
3. Technology: software routers
4. Applicability
5. Q&A

Hardware vs. Software Routers

Why Software Routers ?

Why Software Routers ?

- In the beginning, there was only routing „in software“ – matched required/available transport link speeds
- Some years later, CPUs simply weren't fast enough anymore to „push packets“ reasonably
- Hardware routing with special chipsets was thus the only option for service providers
 - Quite some rounds of development for routing chipsets over time
 - Vendor specific chipsets vs. commercial, off-the-shelf (COTS) chipsets
 - Always a trade between price and performance
- This has changed recently: today's CPUs are powerful enough again for commonly used interface speeds
- „High-enough-speed“ network interface cards are available for x86-based servers
- Not everyone anymore also needs the biggest available interfaces in a router (40/100G ?)

Why Software Routers ? (cont'd)

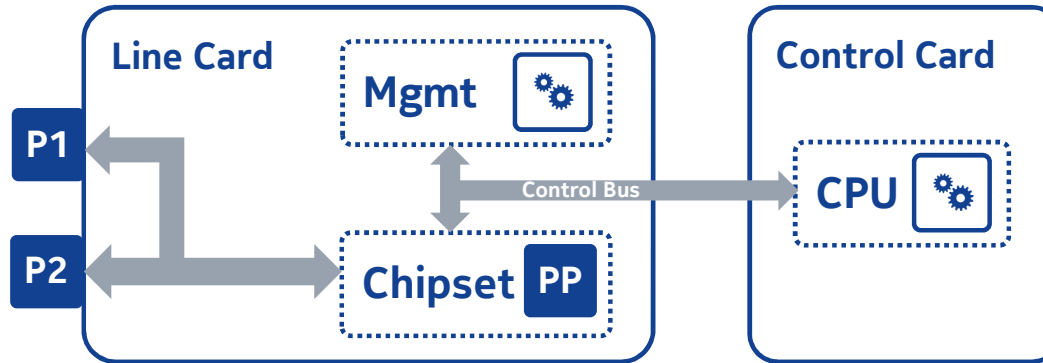
- Hardware routers are special gear, and thus somewhat inflexible in their applicability
- Today's hardware routing chipsets might be „too big“ already for standard enterprise routing (=Internet-routing only, low number of 10G interfaces and some level of BGP)
- Hardware routers do have a bit of cost
- You either need stock, or there might be delivery times

- Can a network operator gain more flexibility and save on costs at the same time by going software-only ?

Hardware vs. Software Routers

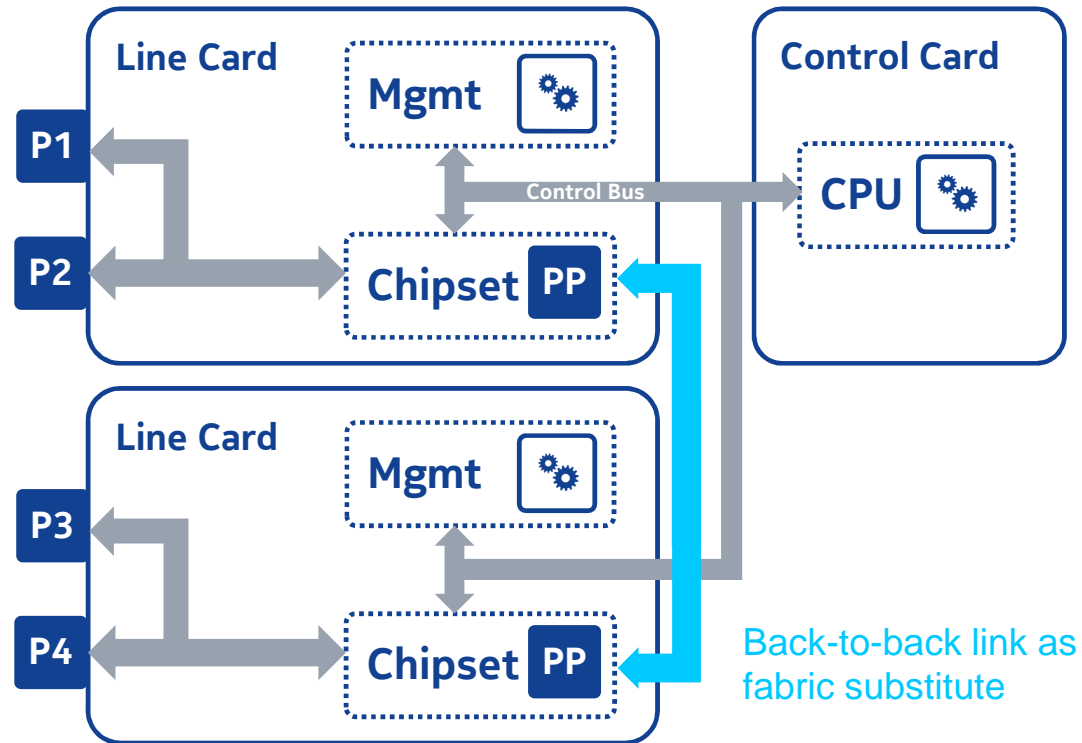
Hardware Routers

Hardware router flavours: a very simple, fabric-less design



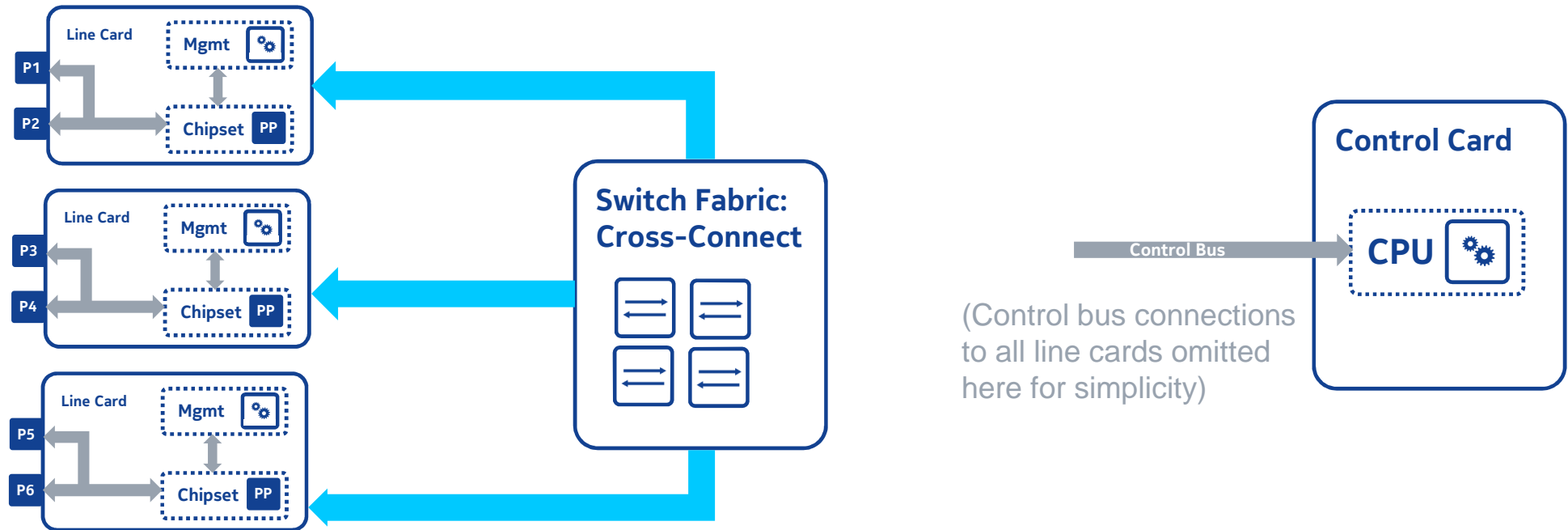
- One control card, and only one single line card
- Control card: runs protocols, keeps lists, tables, state, etc.
- Line card: does all packet processing - lookups, filtering, queueing, forwarding,...
- All traffic stays local within the single line card once it has passed through the ports facing the external world

Hardware router flavours: back-to-back line card design



- One control card, and two line cards
- Traffic must have a sufficient bandwidth path between both line cards – basically 100% of the front port speed as worst-case scenario

Hardware router flavours: full fabric design



- One control card, and many line cards
- Traffic must have a sufficient bandwidth path between any two line cards
- You need a lossless, any-to-any cross-connect as your switch fabric

Hardware vs. Software Routers

Software Routers

From hardware to software: two approaches

- Take a standard OS, and build some routing software on top of it
 - Start with the hardware you have
 - Install the OS of your choice
 - Create/find suitable routing software that runs on this OS, and does all the protocols you need
 - Your OS will automatically provide the data plane for your software router
- Take existing routing software of a hardware router, and port it onto standard hardware
 - Start with the software you already have: ideally already well-proven and hardened
 - Find ways to make it run on standard hardware instead of special hardware
 - See what OS you have to put in between, if applicable
 - Find a way to „provide“ the hardware router's data plane (i.e. hardware chipset) as well...

Option one: standard OS and routing software on top

- Nice, easy, and cheap
- Usually little to no hardware dependencies
- Performance depends on your OS, and it's integrated network stack
- Bear in mind your OS might never have been optimized for packet throughput
- With some tweaks, you might be able to improve the situation here
- Is your OS a real-time OS – do you need things to happen at exact intervals (e.g. BFD) ?
- How do you configure this software router – config files and reboot, or well-known CLI ?
- How does it integrate into your NMS/OSS landscape ?
- Is there support for this setup, do you need it, or can you provide it yourself ?

Option two: port routing software from hardware router

- Effort depends on how that routing software was designed originally
 - If it's x86-based anyway, should be limited effort
 - If it's originally for other CPU flavours, can it be ported ?
- Have the routing software see your server „as close to the real hardware“ as possible
- Where special hardware/chipset functionality is expected, emulate it
- Make sure there is no „stolen CPU cycles“ and the like (real-time behaviour...)
- Apply whatever tricks are necessary to get maximum throughput

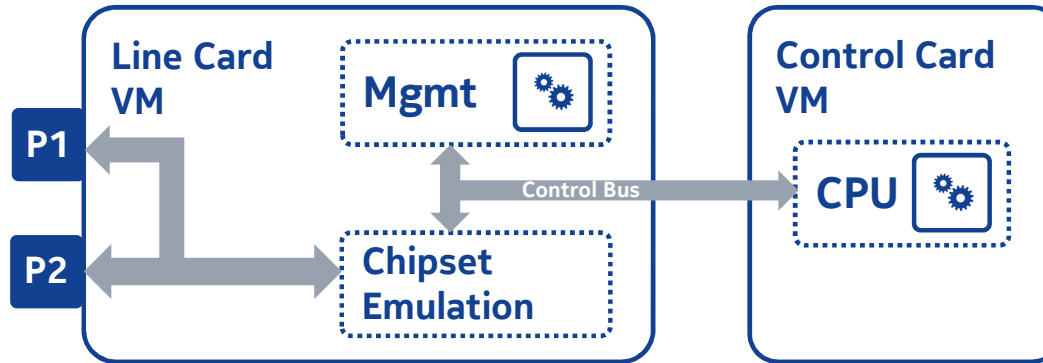
- Use your well-known CLI: same look&feel as regular hardware router
- Supported product: your vendor should be able to help you if required
- There will very likely be a cost with this approach

Option two: hardware router OS - virtualize it !

- Best possible hardware abstraction: use as a VM on a hypervisor host
 - Configure your VM to look as close as possible to what the routing software expects as underlying hardware - use the hypervisor as a hardware abstraction layer
 - Emulate missing hardware in software where required (slow !)
 - Have your VM run a different base OS than your hypervisor
 - Hypervisor OS and services do not need to be exposed into router data plane (security !)

 - However: emulation and abstraction inherently cost performance again...
 - Alternative: have the hypervisor „interfere the least possible way“ for critical items
 - Especially try to avoid the hypervisor’s network stack
 - But then, you might have a hardware dependency again...
- There is no free lunch here either !

Software router virtualized: a very simple, fabric-less design



- One control card, and only one single line card
- Option one: each of them runs as a separate VM on a common hypervisor
- Option two: have one single, shared VM for both functions
- Control bus needs limited bandwidth only
- All traffic stays local within the single line card VM

Software router virtualized: more than one line card VM ?

In reality, multi-line card software routers quickly come to their limits:

- Any line card VM needs to be able to send worst case up to 100% of the front ports capacity to another line card VM
- Even in a two line card system, you need the same additional port capacity for the line card interconnect as you have on the front ports
- But then, why don't you use just ONE line card VM, and ALL available ports „front only“ ?
- „More than two line card systems“ scale even worse in software:
 - You need to emulate the cross-bar switch fabric
 - Either by single point-to-point links → expensive, doesn't scale (see above)
 - Or by a real hardware switch as a fabric emulation
 - But then again, didn't you originally want a software router WITHOUT any special hardware...?

Software routers: how far can you go ?

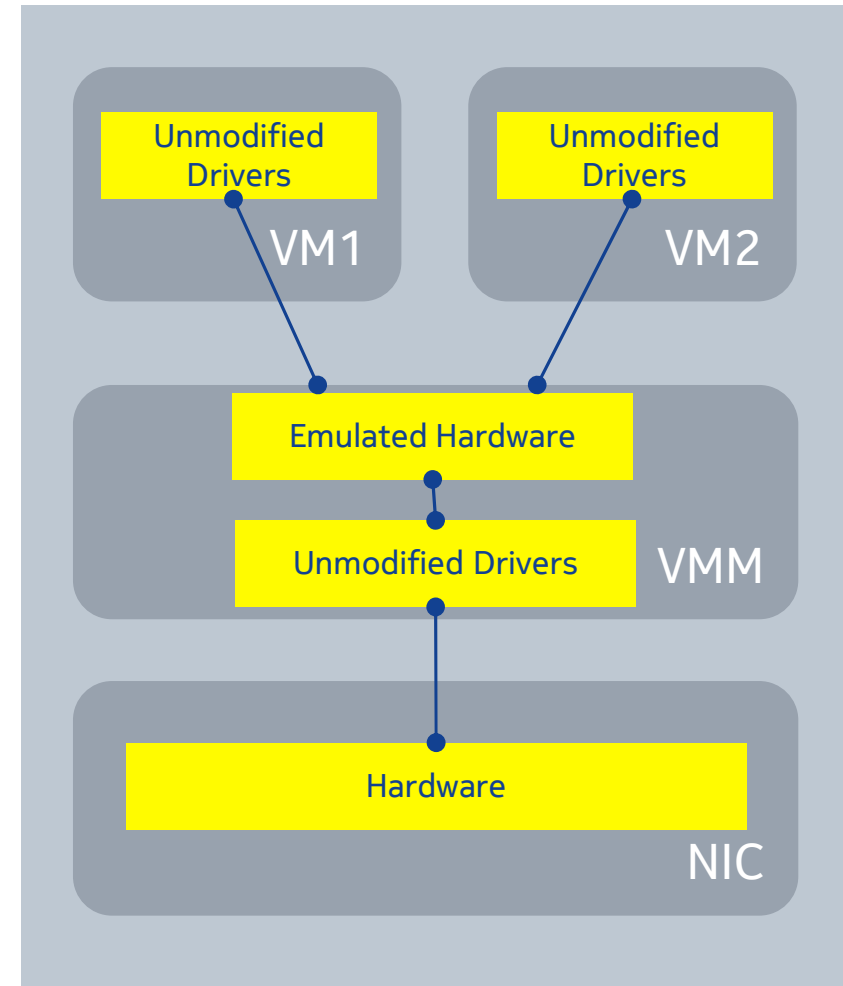
- Remember there was a reason for creating special routing hardware:
 - Highly optimized, built-for-purpose chipsets
 - Special memory types (e.g. TCAM) exist in hardware for a reason – emulation is slow
 - Hardware processing is usually faster than software processing
 - Hardware processing is (should be...) load-independent
 - Assume that where things can be parallelized in hardware, they likely already are
- On a software router, EVERYTHING you do with a packet needs CPU cycles
- The more processing you do on a packet, the more CPU power you need (access lists, Multicast replication,...)
- CPU load goes up with overall throughput

Software routers: how far can you go ? (cont'd)

- You need to get packets in and out of your box as well before you can do anything else
- Your server host only has a limited number of PCI slots
- The NICs you can buy have a limited number of ports
- This puts an upper limit to the number of physical ports you can attach to your software router
- If you use external „fan-out switches“, you have special hardware again...
- How fast can you get packets in and out of your line card VM, using all possible tricks ?

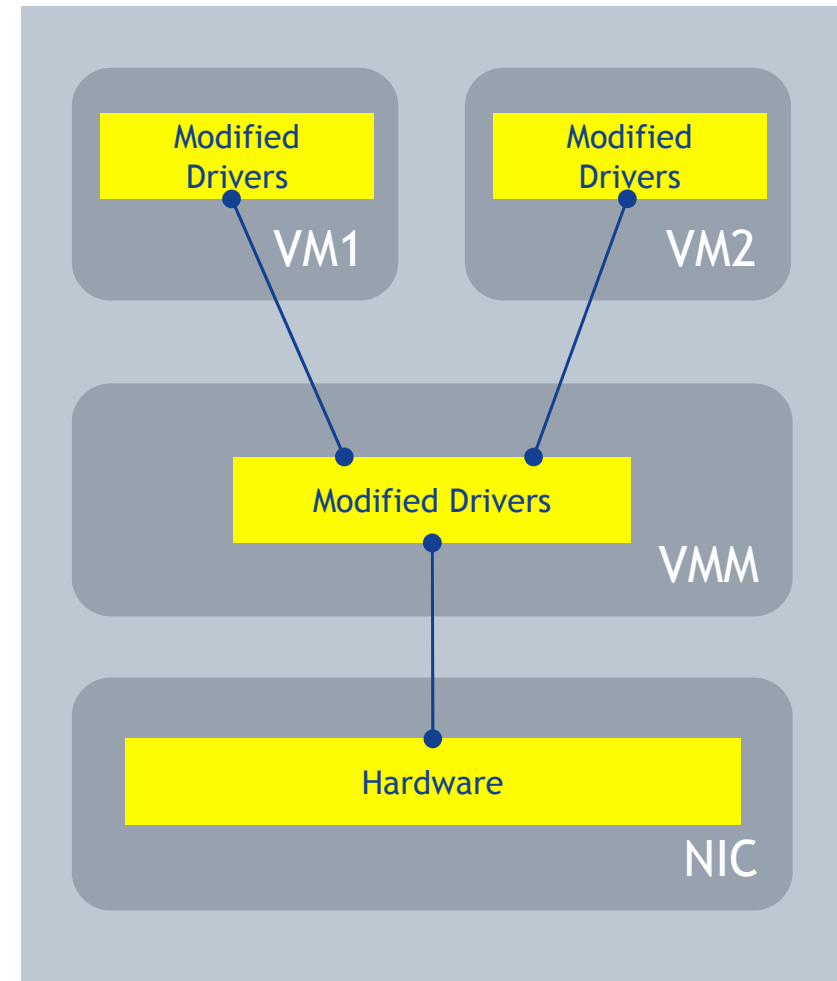
Virtualised I/O: emulated NICs

- **Native** device **drivers** in the guest OS
- VMM **emulates** physical **hardware**
- VMM needs to intercept all traffic and convert it for the physical hardware
- **Very slow** operation
- **Hardware** can be **shared** between VMs
- Doesn't require the guest to have knowledge of the fact that it's virtualised



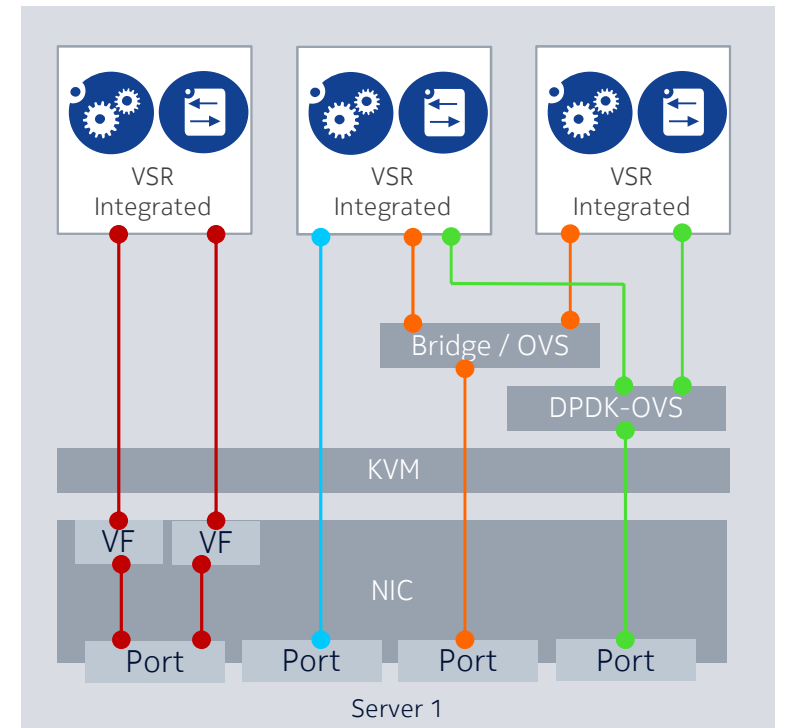
Virtualised I/O: VirtIO NICs

- Para-Virtualised I/O driver
 - The VM is aware of the fact that it is running in an emulated/virtualised environment
- **Requires modifications** to the drivers in both the guest and the host
- Removes the requirement for the hypervisor to have to emulate the hardware
- Guest OS must load the VirtIO driver
- Provides **significantly better performance** than emulated hardware (HVM)



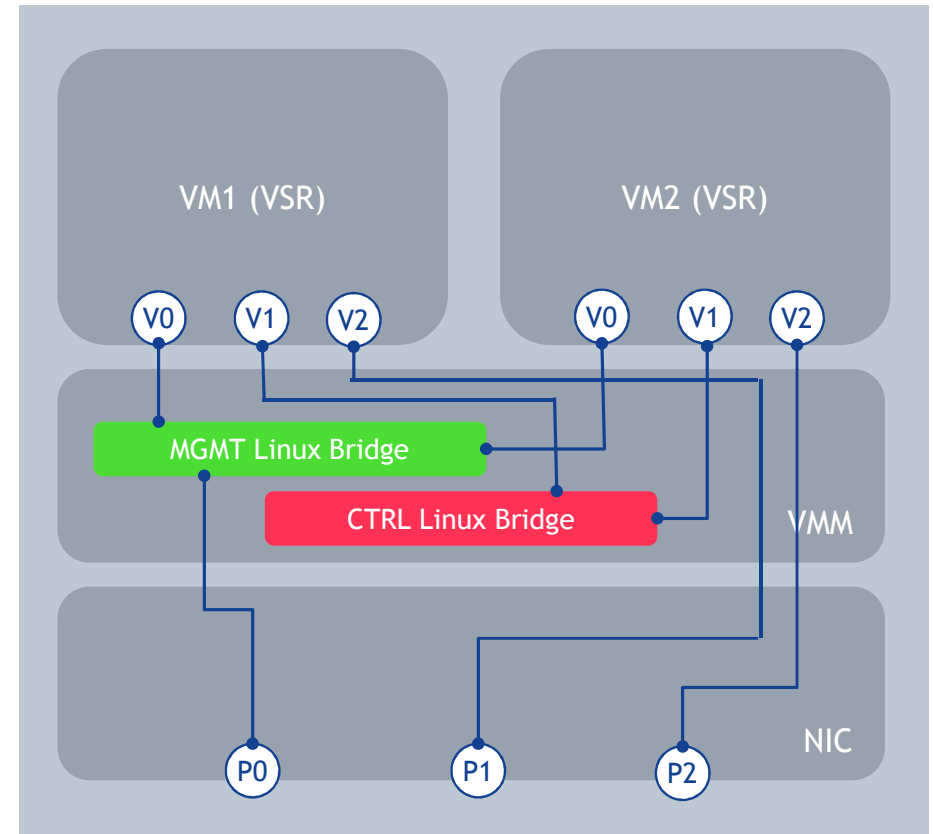
Software routers: virtualised I/O options

- [Linux Bridging](#)
 - Most simple option, available everywhere by default
 - Slowest option of all
- [Openvswitch \(OVS\)](#)
 - Flexible, programmable
- [OVS with DPDK](#)
 - High throughput with flexibility
- [SR-IOV](#)
 - Virtual Functions created in hardware to share ports
 - Highest throughput with some flexibility
- [PCI-Passthrough](#)
 - Highest throughput with most flexibility on the virtual router side, but almost no virtualization stack flexibility on the host side



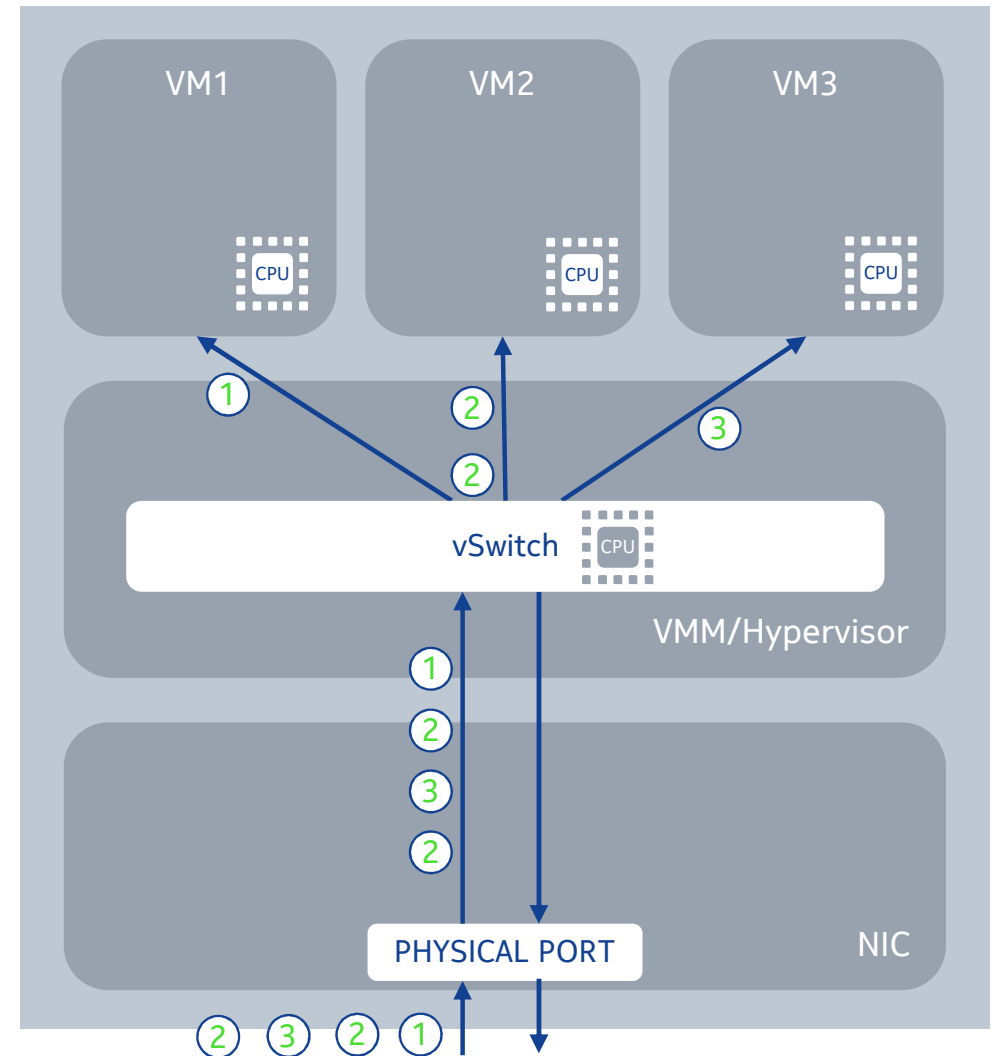
Virtualised I/O: Linux bridging

- **Standard service** available within Linux
- Works within a single physical machine
- The bridge appears like any other Linux Interface
- Multiple virtual and physical interfaces can be connected to it
- Commonly used for the control-interconnects on the VSR within the same physical machine or to share interfaces where **performance is less important**



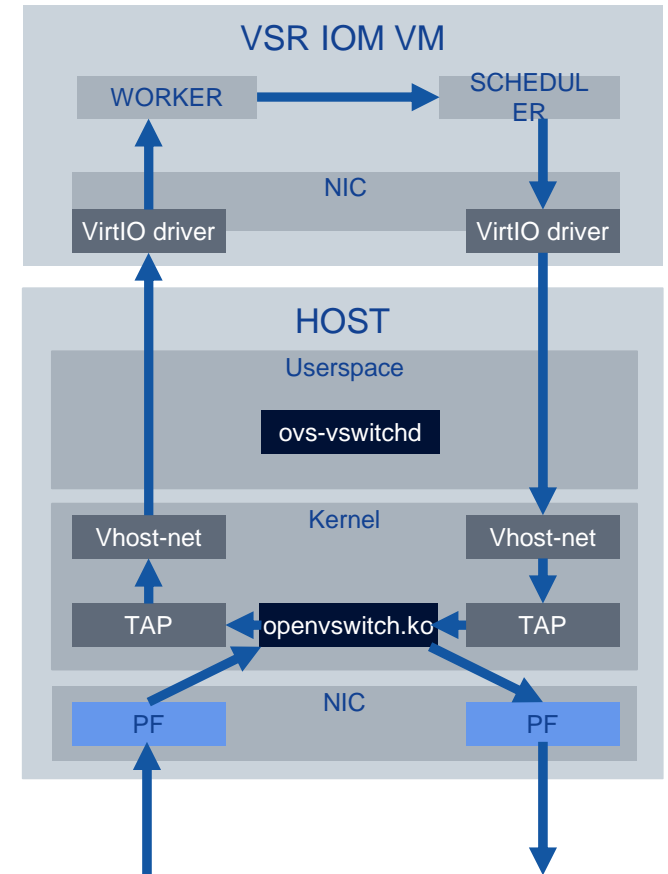
Virtualised I/O: vSwitch

- If no intelligence provided by the NIC a software switch is required in the VMM (VM Manager)
- vSwitch examines each packet and identifies the destination based on the mac address
- vSwitch then directs the packet to the VM
- **Inefficient** as the hypervisor needs to read, understand and deal with each packet
- Ideally: a **single CPU** core **assigned** to deal with the vSwitch
- Interrupts fired for: Packet arriving on NIC, packet being handled by vSwitch, VM dealing with packet



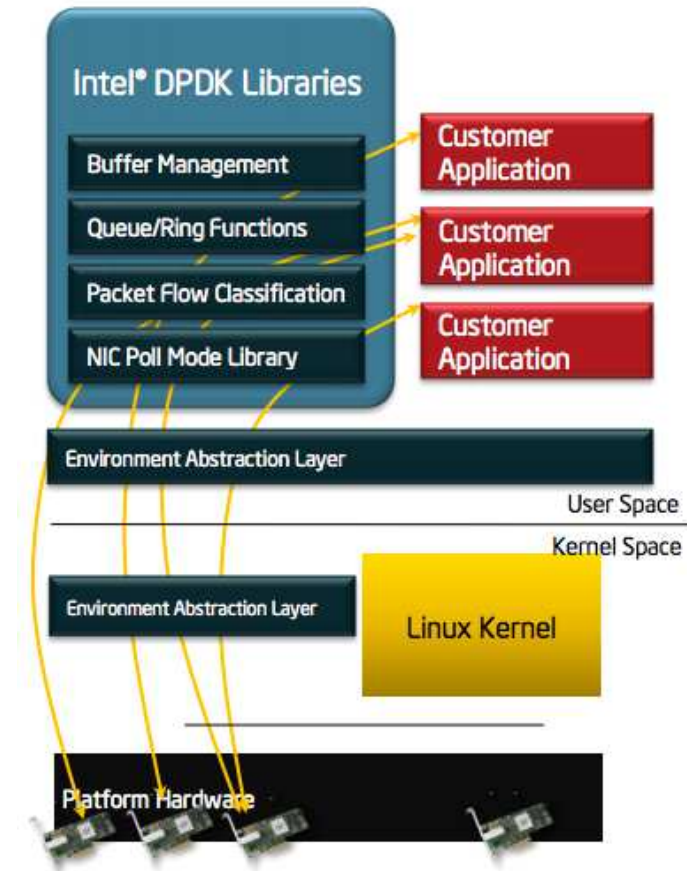
Virtualised I/O: Open vSwitch (OVS)

- Hypervisor connects guest vNIC port to a port of the vSwitch
 - The guest uses a virtualization driver (VirtIO, E1000, VMXNET) for the vNIC port
 - Host implements the other side of the driver in user-space or kernel
- Throughput performance is much lower than pass-through models due to:
 - **Interrupt handling** associated with I/O transfers
 - **Packet copying** between guest and host memory locations
- Possibly relevant for datacenter integration: with Open vSwitch, the packets flowing through the vSwitch can be controlled by Openflow rules for SDN use cases. This includes adding/removing NVO3 encapsulations (VxLAN, GRE, etc.)



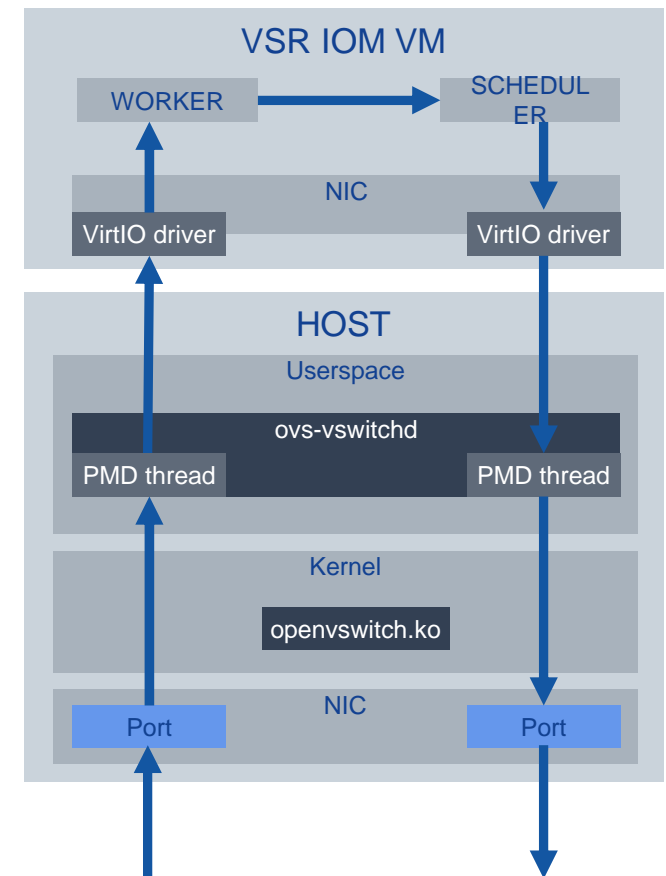
Virtualised I/O: OVS-DPDK

- DPDK is an **open-source toolkit** for **fast packet processing**.
- When OVS is **compiled to use DPDK libraries** and DPDK NIC drivers, the result is a **high-performance vSwitch**, which is referred to as OVS-DPDK (in this document).
- OVS-DPDK is considerably faster (7x to 10x) than native OVS due to the following reasons:
 - The OVS-DPDK fast path moves from the openvswitch.ko kernel module to a **user-space implementation** (the dpif-netdev component of the ovs-vsitchd daemon).
 - OVS-DPDK communicates with virtual machine vNIC ports (that use a VirtIO driver) using **user-space vHost drivers** (vhostuser).
 - **Poll-Mode-Driver** (PMD) threads of the user space ovs-vsitchd process send and receive packets over the OVS switch ports.



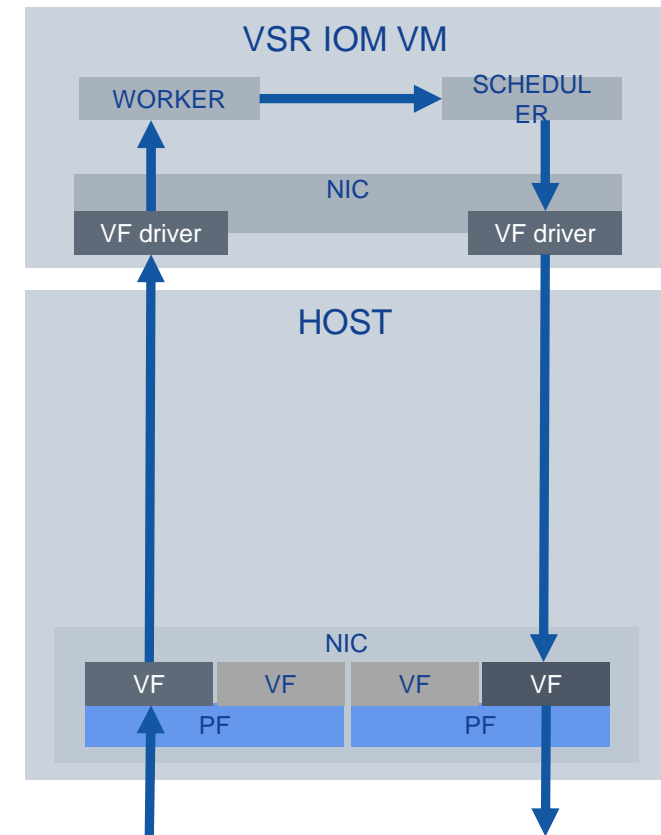
Virtualised I/O: DPDK accelerated OVS

- vSwitch with DPDK acceleration:
 - A software implementation of an L2/L3 switch runs in the host
 - Guests send and receive packets via the vSwitch
 - **Datapath is implemented entirely in user-space**
- A guest vNIC port is logically connected (by the hypervisor) to a port of the vSwitch
 - The guest attaches a VirtIO driver to the vNIC port
 - Host implements the other side of the driver in user-space (vhost-user)
- Throughput performance is significantly higher than native vSwitch model due to use of poll-mode drivers (PMD), more advanced CPU instructions, huge pages, etc.
 - But still lower than pass-through models due to **packet copying** between guest and host memory locations
- Advantages of the native vSwitch model are retained:
 - Live migration/vMotion
 - SDN control by openflow rules
 - NVO3 encapsulations (VxLAN, GRE, etc.)
 - Open Virtual Network (OVN) support



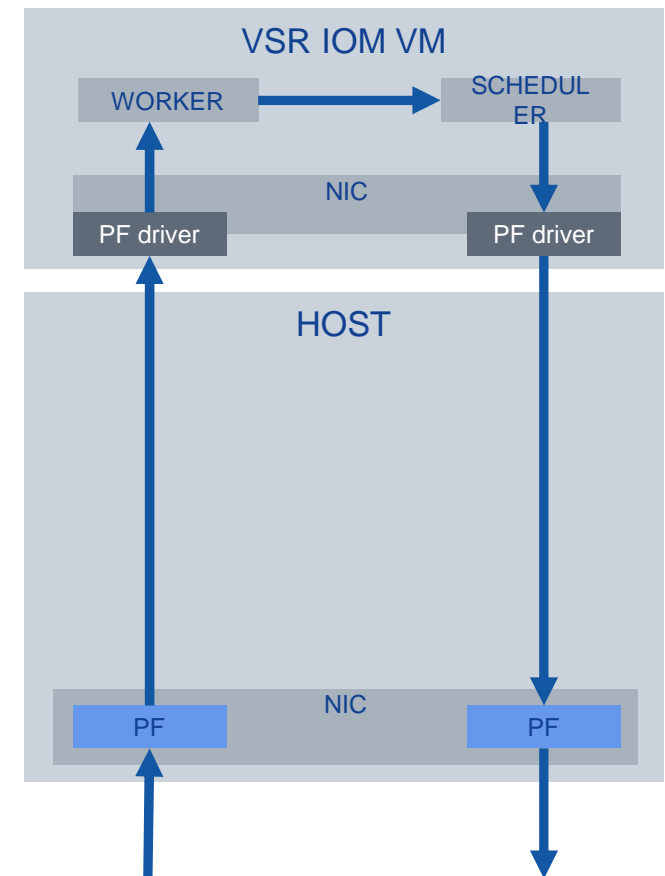
Virtualised I/O: SR-IOV

- Single Root I/O Virtualization
- PCI-SIG standard that allows **one PCIe device** (Physical Function/PF) to appear as **multiple lightweight PCIe devices** (Virtual Functions/VFs)
- vNIC in **guest uses VF driver specific** to the physical NIC type
- Direct I/O path between NIC and VM
 - **Limited hypervisor involvement**
 - VLAN/MAC transparency to be considered on NIC driver side
 - **Zero copy** receive and transmit due to DMA remapping in NIC (guest physical address -> host physical address)
 - **Near-native throughput**



Virtualised I/O: PCI Passthrough

- Allows a **physical PCI device** from the host to be assigned **directly** to a guest
 - Guest controls the port using its own equivalent of the **bare-metal NIC driver**
 - The device is automatically **detached from** the **host OS** drivers when the guest is started and re-attached when the guest shuts down (managed mode)
- Direct I/O path between NIC and VM
 - **No hypervisor involvement**
 - All frames sent through untouched – fully up to the VM to handle
 - **Zero copy receive and transmit** due to DMA remapping in NIC (guest physical address -> host physical address)
 - **Near-native throughput**
- **No sharing of physical NIC** ports by different VMs



Hardware vs. Software Routers

Applicability

Software routers: some very basic rules for applicability

- Great for control-plane intense tasks
 - The control plane of a hardware router is also just some piece of software running on a CPU
 - Running protocols and keeping tables is a nice job for a CPU, and hardly hardware-assisted anyway
- Single line-card designs, reasonably low port count
- Examples:
 - BGP route reflector – very little data plane required
 - CG-NAT appliance – mostly mapping table management, and some header rewriting
 - BNG, subscriber management – keeping state tables, QoS to be emulated
 - L2TP tunnel termination – mainly header management plus some state keeping
 - Large-scale IPsec gateway – encryption can be done in software reasonably well
 - DPI – hardware chipsets usually only look at packet headers anyway
 - Router simulator for lab use (if it actually behaves like real hardware does...)

Software routers: some very basic rules for applicability

- Limited throughput and packet processing activated
 - Good scenario: Enterprise router - low number of 10G interfaces, full BGP table, a bit of peering, otherwise standard routing, little to no QoS, maybe some VPNs in addition
 - Questionable: higher (>4) number of physical 40G/100G interfaces
 - Unrealistic: trying to build a router/switch with many really high bandwidth ports in software
- Do you need guaranteed performance ?
 - For a hardware router, you can get somewhat guaranteed performance values from your vendor
 - Performance you can actually achieve with a software router depends a lot on your specific hardware and software configuration – little to no vendor guarantees for performance possible upfront
 - You will only know how well your actual hardware performs with all your actually configured features once you really try it out on exactly THIS system
 - Any change in server hardware or router configuration WILL affect performance again
- Way out of this: buy a fixed combination of server and routing software as appliance

Software routers: some very basic rules for applicability (cont'd)

- Is your network fully automated already - is compute power a „consumable resource“ ?
 - Software routers can be instantly deployed and destroyed „on-demand“
 - No special hardware to be bought
 - No delivery times from your supplier
 - Use your existing automation platform also for routing, like you do for plain compute-VMs
 - Your datacenter is also your backbone room in this case
- No chance to get to this level of flexibility with regular hardware routers - software routers are WAY more flexible for this scenario
- Cover the possible limited throughput of software routers by more instances in parallel
- „Deploy as you grow“ model - „from pets to cattle“
- If you really need a very big box in one place, you can still add a hardware router there

Software routers: free software or commercial products ?

- Do you need every bit of performance squeezed out ?
 - If so, does your free implementation „pull all the available registers“ ?
 - Or is it simply already „good enough“ for what you have in mind ?
- Do you feel comfortable with the user interface of your routing software ?
- Do you need SLA-based support ?
 - Do you have SLAs towards your own customers that you need to keep ?
 - Does „the community“ always (!) react quickly enough for you ?
 - If not, can you fix really everything just by yourself quickly enough ?
 - Is the time you spend on this also „free“ ?
 - What other things of your regular job can't you do while you fix routing software bugs ?
 - Does your company have someone else with the same skills next to you, just in case...?
- Is your software router for lab use only ?

Q&A

NOKIA

NOKIA

Copyright and confidentiality

The contents of this document are proprietary and confidential property of Nokia. This document is provided subject to confidentiality obligations of the applicable agreement(s).

This document is intended for use of Nokia's customers and collaborators only for the purpose for which this document is submitted by Nokia. No part of this document may be reproduced or made available to the public or to any third party in any form or means without the prior written permission of Nokia. This document is to be used by properly trained professional personnel. Any use of the contents in this document is limited strictly to the use(s) specifically created in the applicable agreement(s) under which the document is submitted. The user of this document may voluntarily provide suggestions, comments or other feedback to Nokia in respect of the contents of this document ("Feedback").

Such Feedback may be used in Nokia products and related specifications or other documentation. Accordingly, if the user of this document gives Nokia Feedback on the contents of this document, Nokia may freely use, disclose, reproduce, license, distribute and otherwise commercialize the feedback in any Nokia product, technology, service, specification or other documentation.

Nokia operates a policy of ongoing development. Nokia reserves the right to make changes and improvements to any of the products and/or services described in this document or withdraw this document at any time without prior notice.

The contents of this document are provided "as is". Except as required by applicable law, no warranties of any kind, either express or implied, including, but not limited to, the implied

warranties of merchantability and fitness for a particular purpose, are made in relation to the accuracy, reliability or contents of this document. NOKIA SHALL NOT BE RESPONSIBLE IN ANY EVENT FOR ERRORS IN THIS DOCUMENT or for any loss of data or income or any special, incidental, consequential, indirect or direct damages howsoever caused, that might arise from the use of this document or any contents of this document.

This document and the product(s) it describes are protected by copyright according to the applicable laws.

Nokia is a registered trademark of Nokia Corporation. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.