Sharif University of Technology
Computer Engineering Department

Software-Defined Networking

Ali Movaghar
Mohammad Hosseini

Network Functions Virtualization

TA: Iman Rahmati & Farbod Shahinfar

Includes slides from a course taught by Umakishore Ramachandran (Georgia Institute of Technology)

What Are Network Functions (NF)?

- Firewall
  - Filters traffic based in pre-defined rules
- Intrusion Detection/Prevention System (IDS, IPS, DPI, DLP)
  - Perform complicated analysis of traffic to detect attack/suspicious activity
- Network Address Translation (NAT, CG-NAT)
  - Translates private IP address space to public IP address space and vice versa
- Load Balancer
  - Distribute traffic to a pool of back-end servers
- Wan and Application Optimizers (Cache server, CDN, HTTP Proxy)
  - Reduce bandwidth consumption using techniques such as caching, traffic compression
- Tunneling Gateway (IPSec/SSL accelerator, VPN)
  - Provides abstraction of the same IP address space for networks that are physically separated
  - Multiple sites communicate over WAN using tunnels between gateways
- QoS
  - Service assurance, SLA monitoring
- Switching (Ethernet switch, Router)
- Mobile Network Nodes (HLR, SGSN, GGSN, CGSN)

Middlebox

- ❖ Standalone hardware boxes (network appliances) providing specific network functions
  - ➢ Vendor-proprietary
  - ➢ Optimized for a specific function
  - ➢ High-performance and high-throughput

Middlebox Issues

- Expensive to procure and maintain
- Launching a new network service needs to find space and power to accommodate the new boxes.
- Hardware-based appliances rapidly reach end of life
- Issues related to proprietary devices and vendor-specific APIs
  - Complex to operate and manage
  - Difficult to automate and integrate into a high-level orchestration platform
- Vendor lock-in
  - Difficult and expensive to migrate to a different solution
- Failures of middleboxes lead to network outages
- High capital and operational expenditure
  - Provisioning is done based in the peak capacity
    - Typically very underutilized
  - Management/Maintenance cost is high

Solution

- Replace middleboxes by software entities
- Run such network functions as an “application” on **general-purpose (COTS) servers**
- **Benefits**:
  - Low cost of deployment
  - Better resource utilization
  - Scaling is easily possible (lower CAPEX)
  - It is possible to easily switch between vendors
  - Failures are easier to deal

Examples of Software Middleboxes

- Software middleboxes have been ongoing for a long time
  - **Nginx** (Web reverse proxy, load balancer, cache)
  - **Softether**, **OpenVPN**
  - **Zeek**, **Snort** (Intrusion detection and prevention)
  - **Linux iptables** (provides NAT and firewall)
  - **Open vSwitch**
- They do not require special hardware.

Fundamental Components of Software Middleboxes

❖ What is the fundamental component for developing a software middlebox?

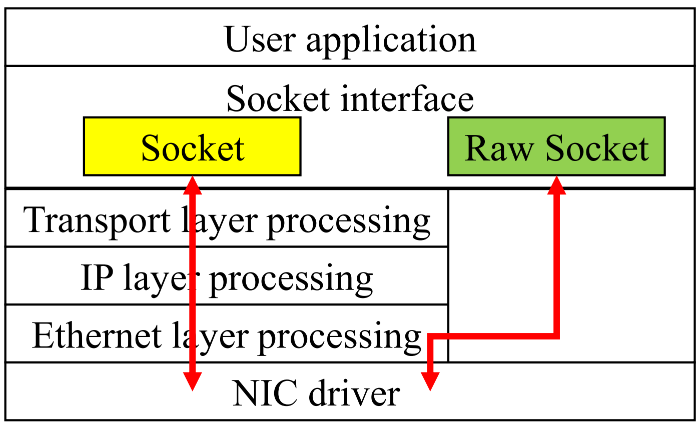
➢ Unix **Sockets**

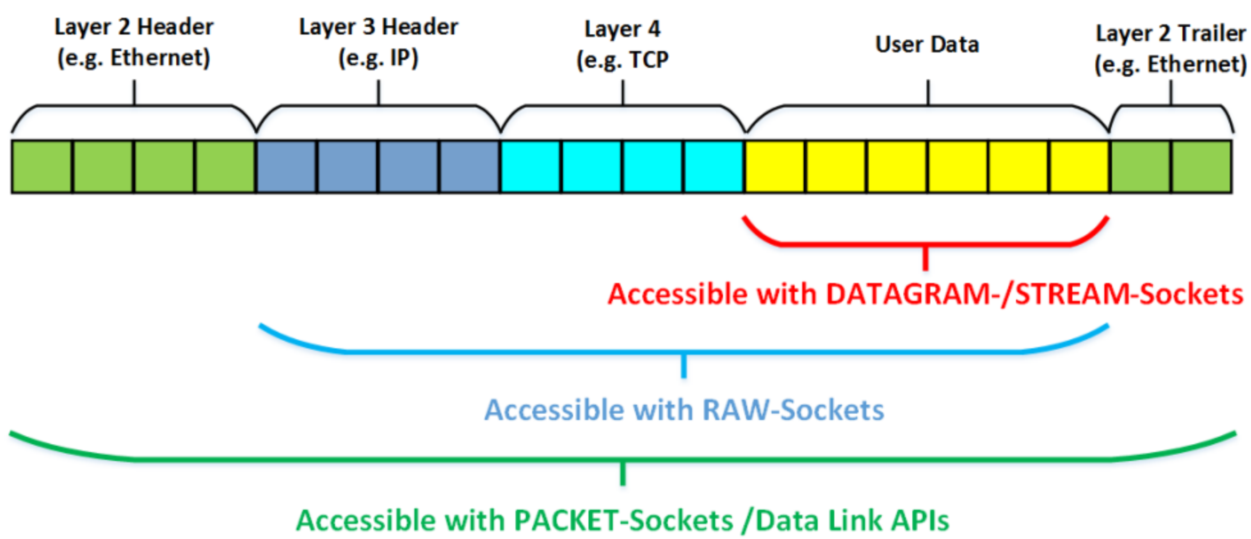
- System calls *read()* and *write()* to Linux kernel for reading and writing to a socket

Fundamental Components of Software Middleboxes

- What if we did not want to work in the application layer?

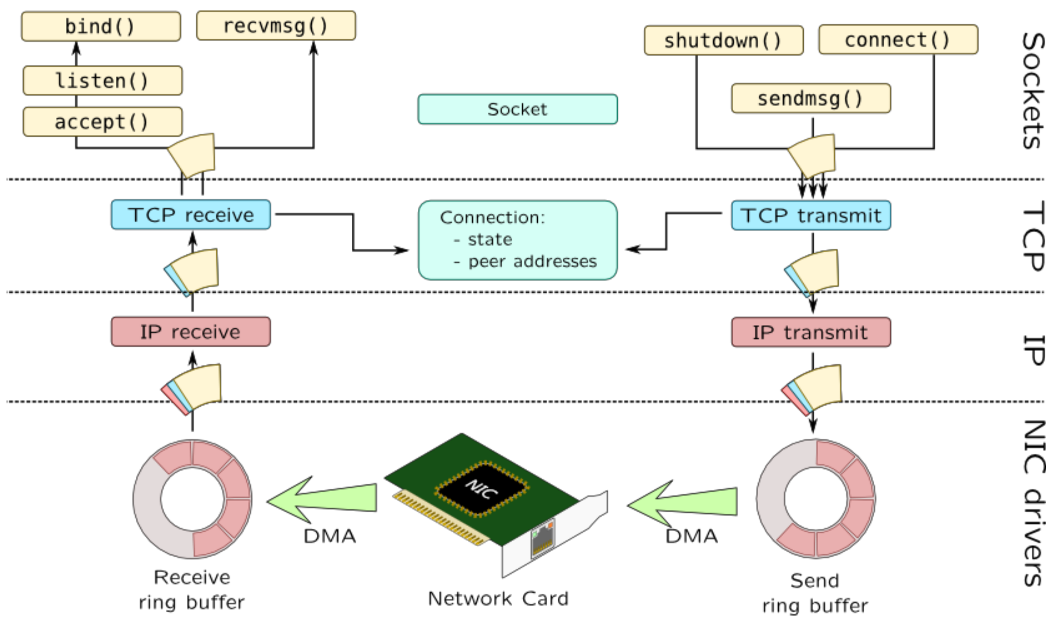
- Raw Linux sockets enable developer to read/write raw bytes (MAC layer data) from/to NIC





What happens when a packet arrives

- NIC uses Direct Memory Access to write the incoming packet to the memory (ring buffer)
- NIC generates an interrupt which is delivered to the OS by the CPU
- OS handles the interrupt, allocates kernel buffer and copies the packet (from the ring buffer) into the buffer for IP and TCP processing (depending the particular protocol that is being used)
- After protocol processing, packet payload is copied to the application buffer (user-space) for processing by the application



Approach to Software Middlebox Deployment

- The software middleboxes can run on **bare metal** but we can also think about **virtualizing**

- Using a VM for hosting a network function (instead of running on bare metal servers)

  - Better portability (all dependencies are inside the VM image)
  - Platform agnostic
  - Consolidate equipment types (reducing power/space)
  - Each NF instance is shielded from software faults from other network services

Network Functions Virtualization

- Network functions virtualization (NFV) is a way to virtualize network services, such as routers, firewalls, and load balancers, that have traditionally been run on proprietary hardware.

- These services are packaged as virtual machines (VMs) on **commodity hardware**, which allows service providers to run their network on standard servers instead of proprietary ones.

- The separation of network services from dedicated hardware means network operations can provide new services dynamically and without installing new hardware.

- The concept of NFV was initiated by a white paper published by the European Telecommunications Standards Institute (ETSI) consortium in “SDN and OpenFlow world congress” (2012).

Network Functions Virtualization

Why was the concept of software middlebox and NFV introduced so late?

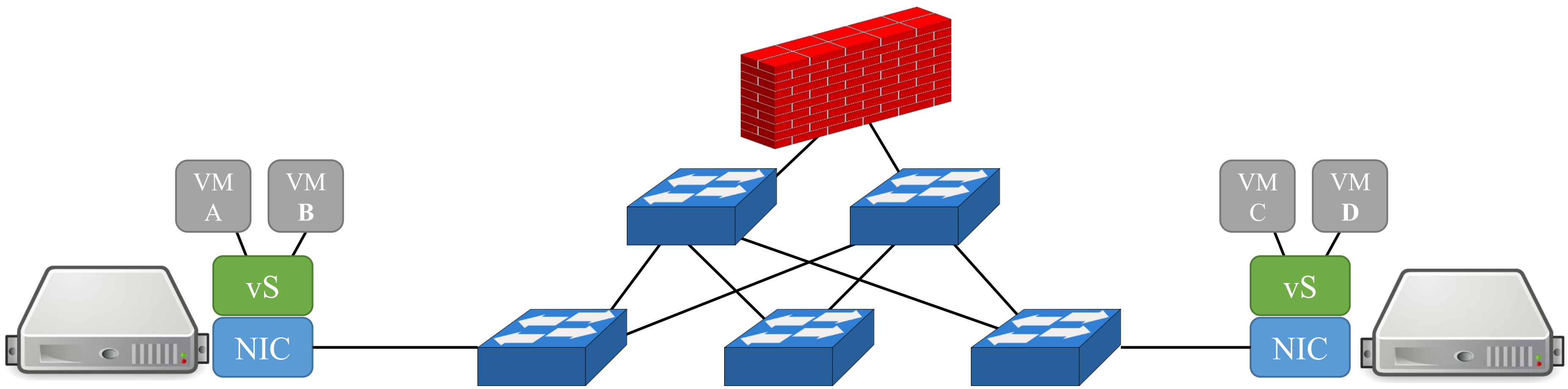
Network Functions Virtualization

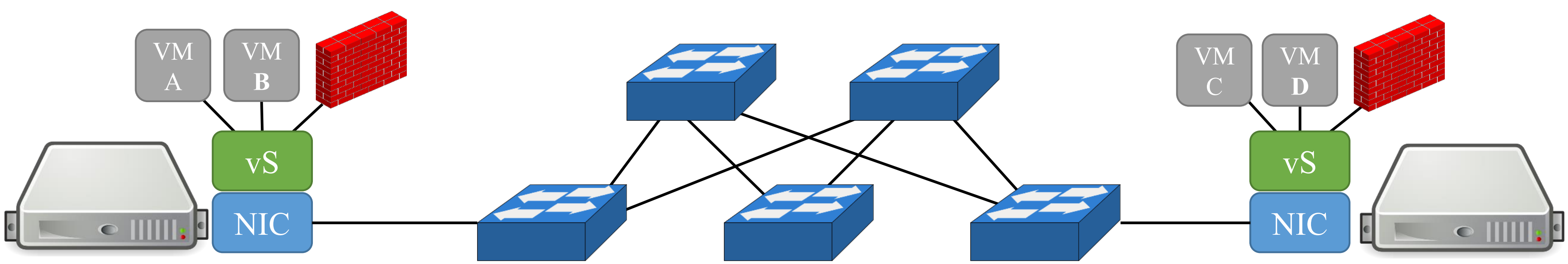
The virtualization technology, cloud, SDN, and their programmable automation tools pave the way for this new paradigm of deploying network functions

Network Functions Virtualization

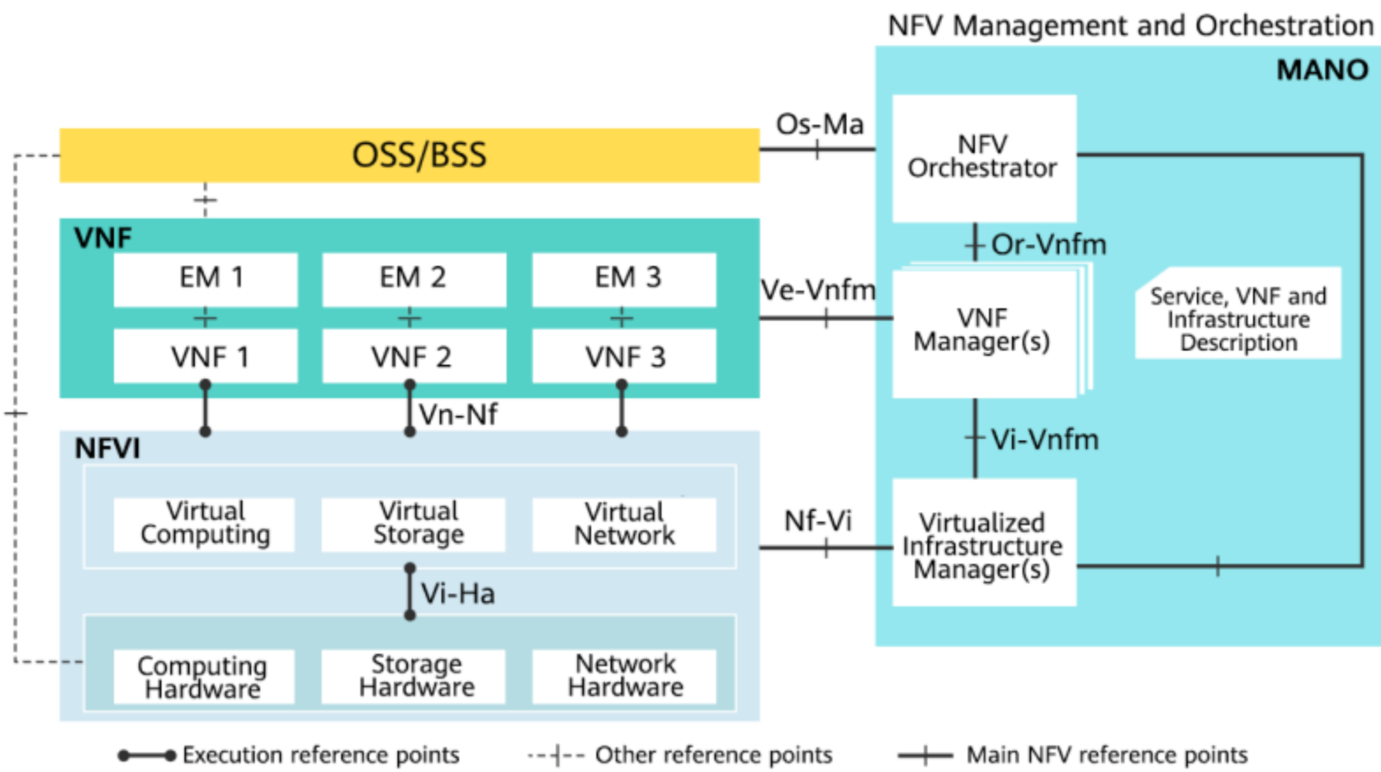
- Virtual appliances running on VMs can use the same automation tools for deployment as VM applications do for servers.
- This allows the same cloud management tools that move and track all the virtual sessions to manage and track the network services (functions) associated with the applications.
- In other words, self-services workflows can be built that deploy servers/applications and network services together, as a single deployment.
- In this model, network functions become another set of virtual appliances, and as such they can easily scale up and down as needed by the application.
- They can also be turned up or turned off easily.

An Example





NFV Framework

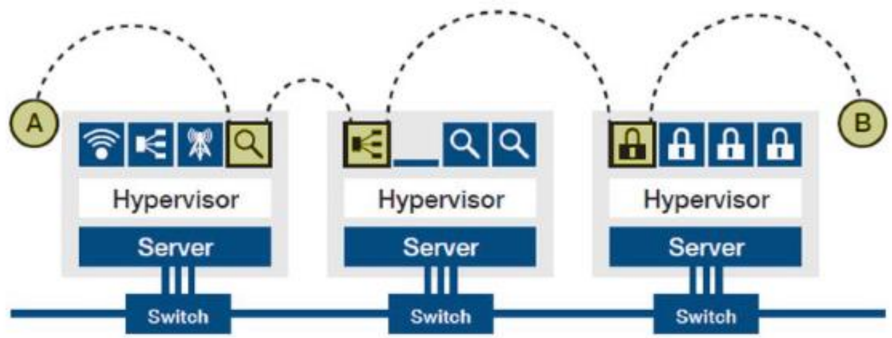


NFV Concepts

- Virtualized Network Function (VNF): Software implementation of NF that can be deployed in a virtualized infrastructure.
- NFV Infrastructure (NFVI): Hardware and software required to deploy, manage and execute VNFs, including computation, networking, and storage.
- Virtualized Infrastructure Manager (VIM): Management of computing, storage, network, and software resources
- VNF Manager: VNF lifecycle management. E.g. instantiation, update, scaling, query, monitoring, fault diagnosis, termination.
- NFV Orchestrator: Automates the deployment, operation, management, coordination of VNFs and NFVI.

NFV Concepts

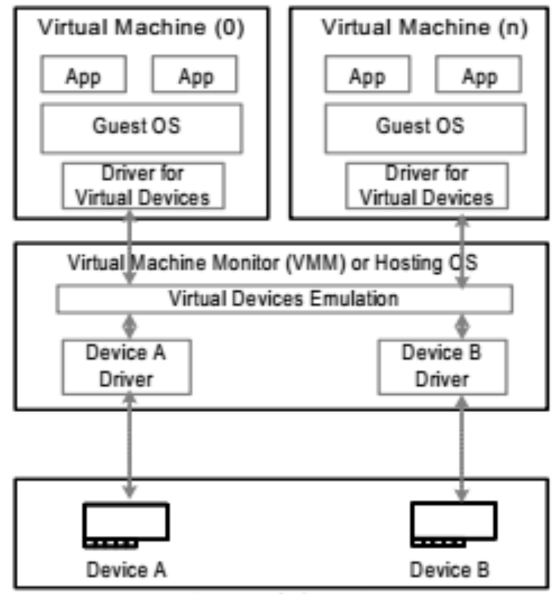
- VNF Forwarding Graph: Service chain when network connectivity order is important, e.g. monitoring, firewall, NAT, load balancer.



NFV Performance Issues

Trap-and-Emulate

- "trap-and-emulate" technique of a hypervisor carries out privileged operations of a VM which is running in user mode.
- I/O is performed via system calls
- When guest VM performs I/O operation
  - Executes system call
  - Guest kernel is context switched in
  - Privileged instructions are invoked for reading/writing to I/O device
- But guest kernel is actually running in user-space
  - Guest VM is a user-space program from the host's perspective
  - Execution of privileged instructions by a user-space program results in a trap
- Trap is caught by the hypervisor
  - Performs the I/O on behalf of the guest VM
  - Notifies the guest VM after I/O operation finishes

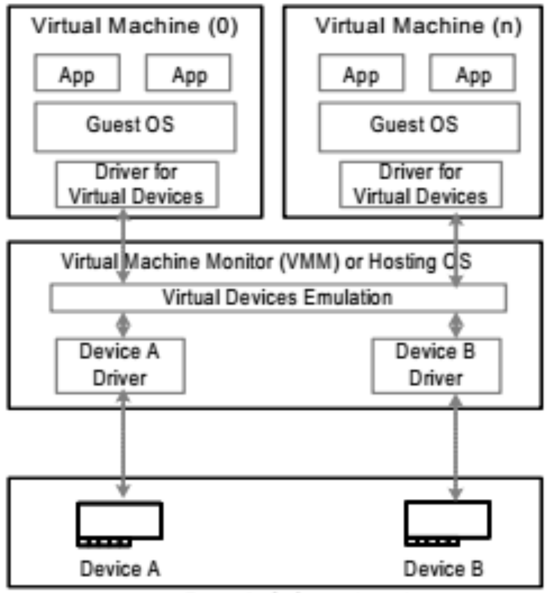


Example Software-based I/O Virtualization

Downsides of Trap-and-Emulate for NF

- Host kernel has to be context switched in by the hypervisor to activate the network device driver and access the hardware NIC
- Duplication of work by the virtual device driver in the guest VM and the actual device driver in the host

- NF incurs the above overheads
  - For each packet that is sent to the NIC
  - For each packet that is received from the NIC



Example Software-based I/O Virtualization

Eliminating the Overhead of Virtualization for NF

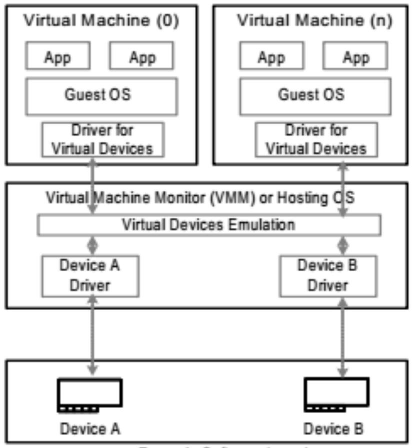
- Two approaches to eliminate I/O virtualization overheads
  - Intel VT-d
  - SR-IOV

Intel VT-d

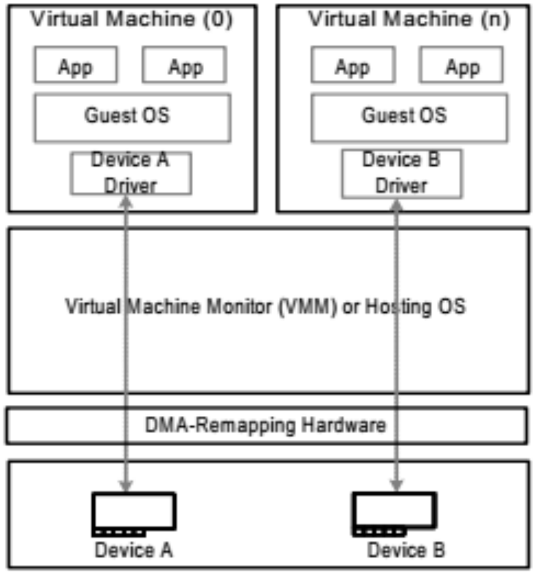
❖ Intel Virtualization Technology for Directed I/O (VT-d)

❖ Avoids overheads of trap-emulate for every I/O access
- ➢ Allows remapping of DMA regions to guest memory
- ➢ Allows interrupt remapping to guest's interrupt handler
- ➢ Configuration registers are mapped to guest VM's memory for memory-mapped IO

❖ Effectively **direct access for guest machine** to I/O device hardware
- ➢ The I/O device (e.g. NIC) is owned by the guest VM



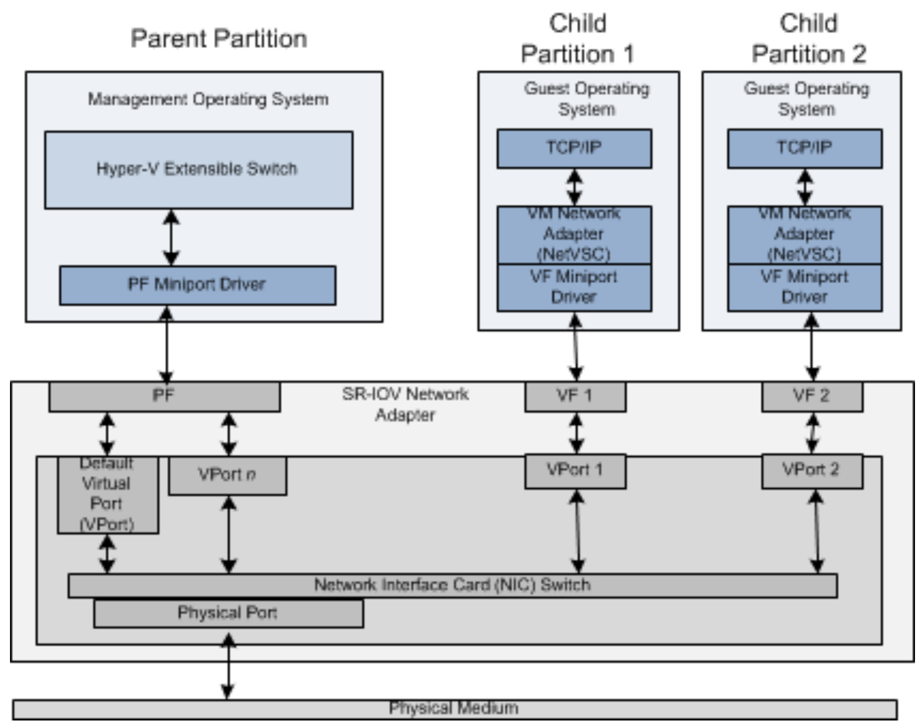
Example Software-based I/O Virtualization



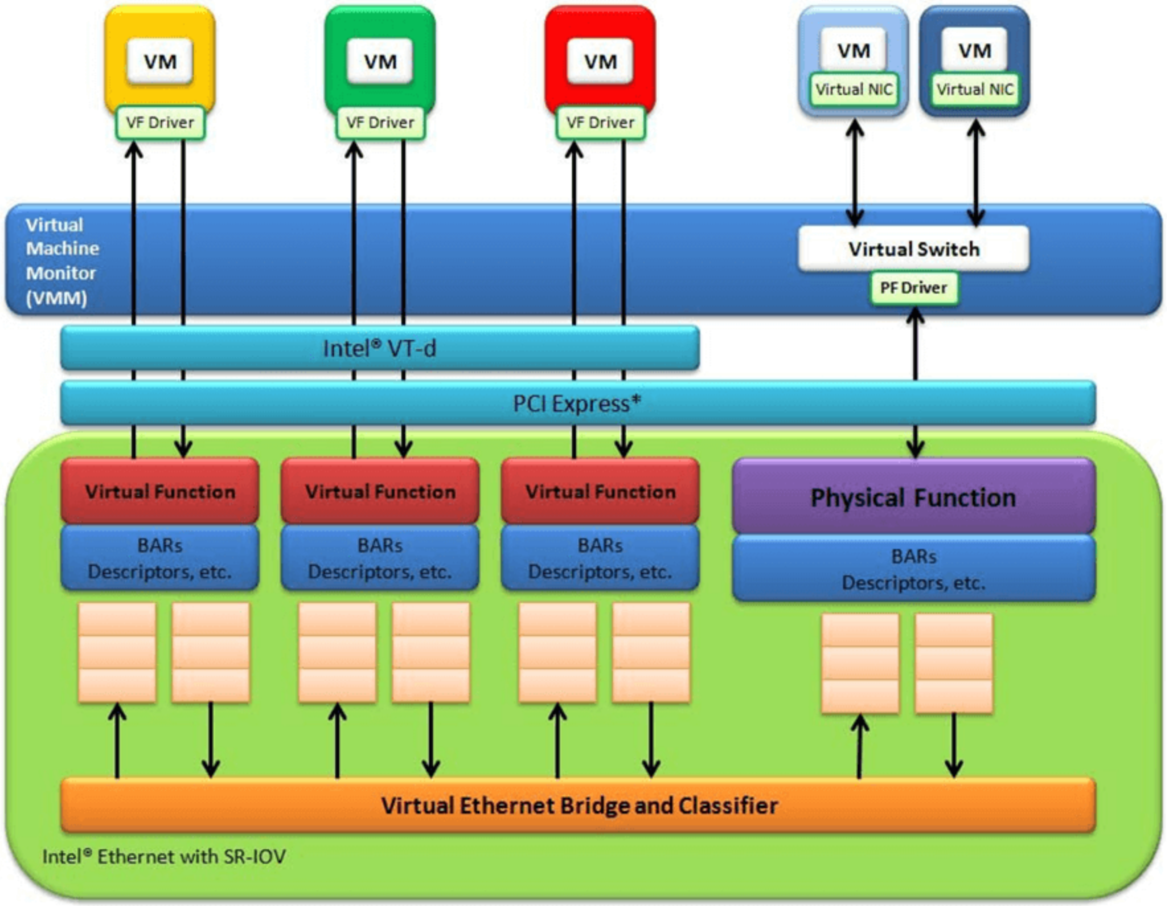
Direct Assignment of I/O Devices

SR-IOV

- Single Root I/O Virtualization (SR-IOV)
- Multiple VMs need to access the same device (e.g. NIC)
- SR-IOV is an extension of the PCIe specification
- Each PCIe device (physical function) is presented as a collection of virtual functions
- Separates the configuration register space for each virtual function
- Each virtual function can be assigned to a different VM
- Allow performance isolation



SR-IOV + VT-d



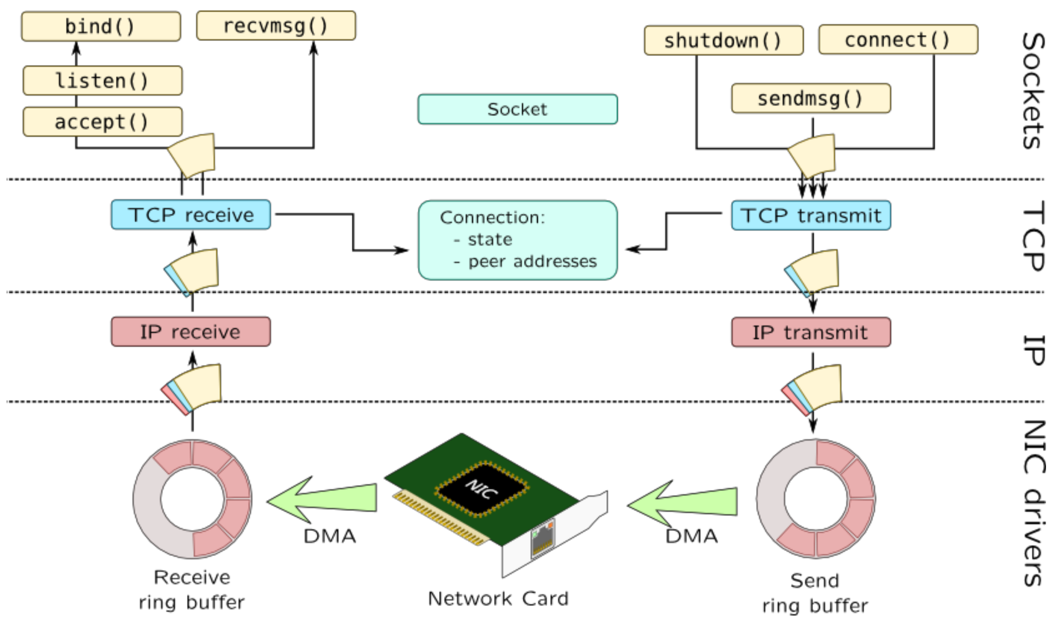
VM
VF Driver
VM
VF Driver
VM
VF Driver
VM
Virtual NIC
VM
Virtual NIC
Virtual Machine Monitor (VMM)
Virtual Switch
PF Driver
Intel® VT-d
PCI Express*
Virtual Function
BARs Descriptors, etc.
Virtual Function
BARs Descriptors, etc.
Virtual Function
BARs Descriptors, etc.
Physical Function
BARs Descriptors, etc.
Virtual Ethernet Bridge and Classifier
Intel® Ethernet with SR-IOV

The Bottleneck of the Guest Kernel

- Intel VT-d provides the means to bypass the hypervisor and go directly to the VM (i.e. the guest kernel which is usually Linux).
  - NF is the application on top of the VM
- However, kernel presents a new bottleneck
  - Specifically for NF applications
  - Sole purpose of NF applications is to read/write packets from/to NICs (packet processing)

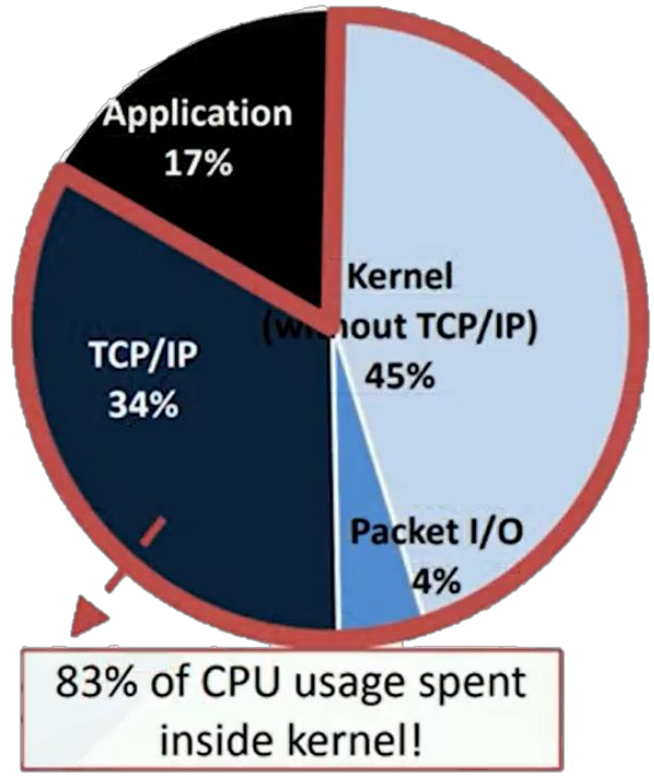
Recapping Packet Processing

- NIC uses Direct Memory Access to write the incoming packet to the memory (ring buffer)
- NIC generates an interrupt which is delivered to the OS by the CPU
- OS handles the interrupt, allocates kernel buffer and copies the packet (from the ring buffer) into the buffer for IP and TCP processing (depending the particular protocol that is being used)
- After protocol processing, packet payload is copied to the application buffer (user-space) for processing by the application



CPU Usage of an Example Networking Application

- ❖ A web server on Linux
  - ➢ Web server (Lighttpd) serving a 64 byte file



Source: “mtcp a highly scalable user-level tcp stack for multicore systems”, (NSDI 2014)

Performance Hits

- One interrupt for each incoming packet
- Dynamic memory allocation (packet buffer) on a per packet basis
- Interrupt service time
- Context switch to kernel and then to the application implementing the NF
- Copying packets multiple times
  - From DMA buffer to kernel buffer
  - Kernel buffer to user-space application buffer
  - A network functions may or may not need TCP/IP protocol stack traversal

We need a solution to reduce the impact of traversing the kernel stack

Solution

❖ **Bypassing the Linux kernel**

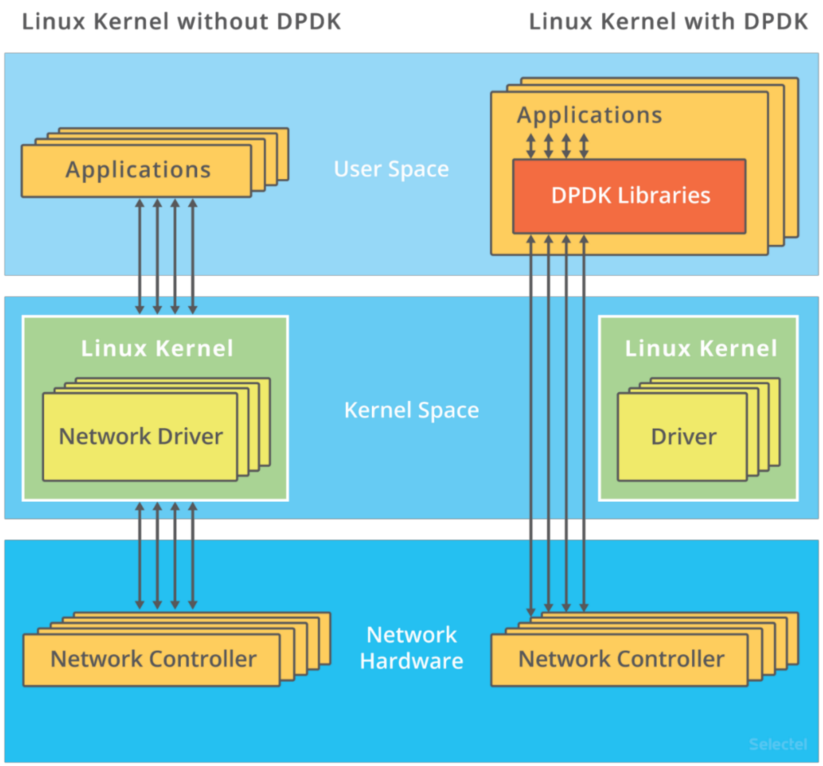
➢ DPDK (Data Plane Development Kit)

- A set of data plane libraries and NIC drivers for offloading packet processing from the OS kernel to processes running in user space
- Created by Intel in 2010
- Now, an open-source project under the Linux Foundation

- Relies on **polling** to read packets instead of interrupts
- **Pre-allocates** buffers for packets (instead of allocating buffers on the fly)
- **Zero-copy** packet processing
  - NIC uses DMA to write packets into pre-allocated application buffers
- Processes packets in **batches** as opposed to individually

DPDK Features

- Buffers for storing incoming and outgoing packets are in user-space memory
  - Directly accessible by the NIC DMA
- NIC configuration registers are mapped in user-space memory (address space of the application)
  - Can be modified directly by user-space application
- Effectively bypasses the kernel for interacting with NIC
  - There is a very small component in the kernel which is responsible for initialization and setting up memory mappings



Poll Mode driver

- DPDK uses polling-mode drivers instead of the interrupt-driven processing provided in the kernel

- Interrupts on packet arrival are disabled
  - NIC directly transmits from/into the user-space transmit and receive queues
  - We can sample NIC registers to see if there is any packets that have been received

- CPU is always busy polling for packets even if there are no packets to be received

- Receive and transmit can happen in batches to increase efficiency

```
// DPDK pseudocode
// Example NF: Load balancer

while (true) {

   buff = bulk_receive(in_port)

   for pkt in buff {
      out_port = look_up(header(pkt))
      // handle failed lookup
      out_buffs[out_port].append(pkt)
   }

   for out_port in out_ports {
      bulk_transmit(out_buffs[out_port])
   }

}
```

Protocol Processing

- No overhead of copying packet data
- NIC DMA transfers packets directly to user-space buffers

- Protocol processing (TCP/IP) is done using those buffered packets in place if needed by the network function
  - Note: not all NFs require TCP/IP processing

DPDK Other Optimizations

- DPDK can exploit hardware features of NICs and modern CPUs
- DPDK can also address the challenges of multi-core and multi-CPU (NUMA) challenges