OpenNetVM Architecture

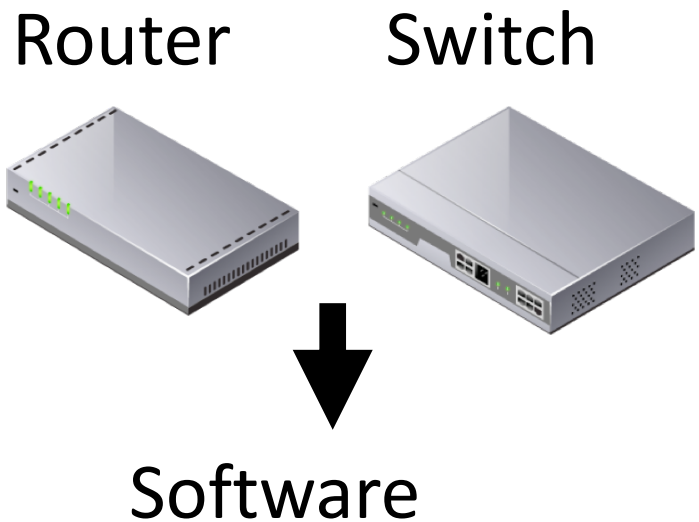
K. K. Ramakrishnan
University of California, Riverside

Joint work with: Timothy Wood (GWU) and many others

UCR

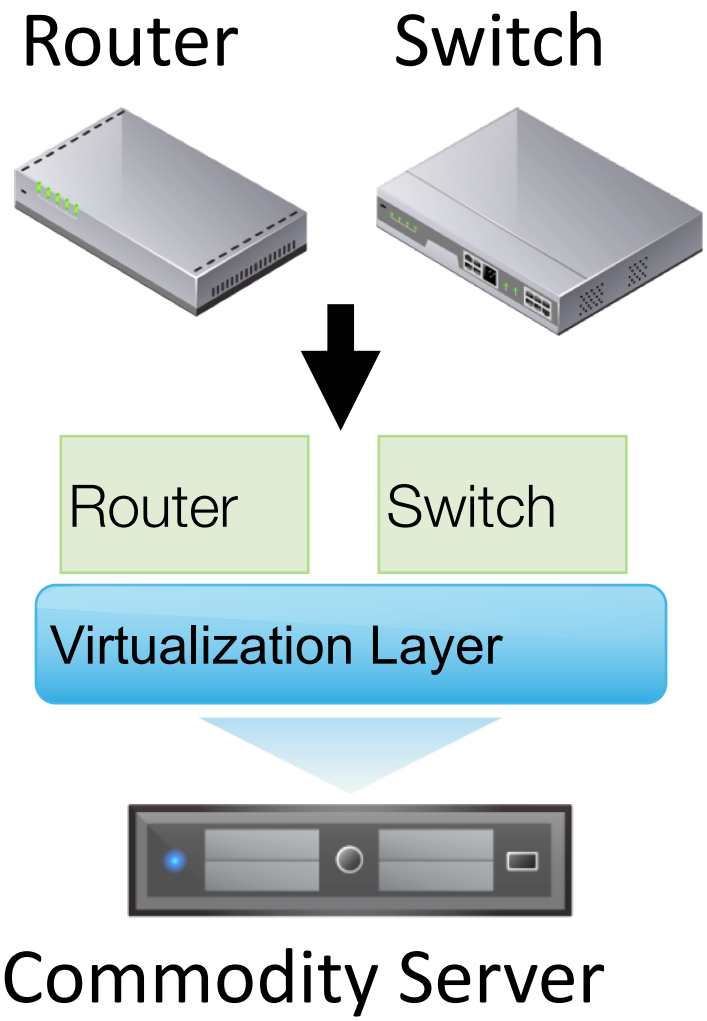
Network Functions in Software

- Make an efficient, customizable **data plane**
  - Services (DNS, DHCP…), firewalls, proxies, IDS, DPI, routers, switches, etc
- Run network functions in software
  - More flexible than hardware
  - Slower than hardware…



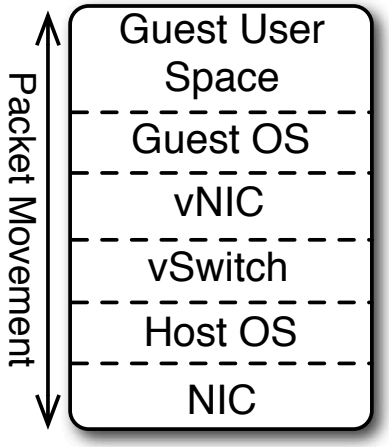
Network Function Virtualization

- Make an efficient, customizable **data plane**
  - routers, switches, firewalls, proxies, IDS, DPI, etc
- Run network functions in software
  - More flexible than hardware
  - Slower than hardware…
- Better yet, run them in virtual machines
  - Isolates functionality, easy to deploy and manage
  - Slower than regular software…



Virtualization Overheads

- **Virtualization layer** provides (resource and performance) isolation among virtual machines
- **Isolation** involves many functions such as access permissions (security), ability to schedule and share etc.
- **Network overhead** (packet delivery) is one of the most critical concerns
- A generic virtualization architecture includes several critical boundaries – host OS, virtual NIC, guest OS, and guest user space–getting packet data there includes **memory copies**



Jinho Hwang, K.K. Ramakrishnan, and Timothy Wood, "NetVM: High Performance and Flexible Networking using Virtualization on Commodity Platforms," NSDI '14.

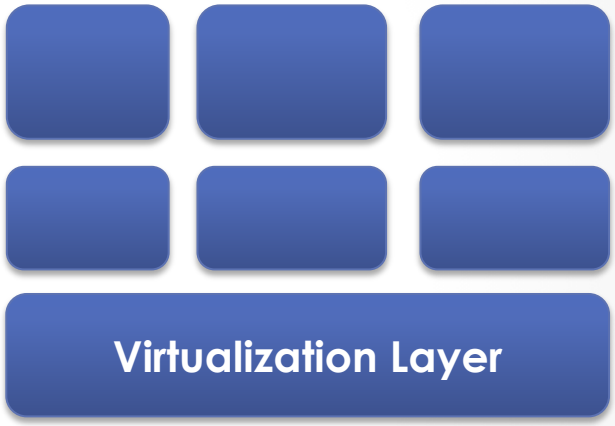
Network Function Virtualization

- Network Function Virtualization (NFV) has its benefits - beyond flexibility

Existing Network Functions

6 Machines : Power x $HW x $SW

NFV-enabled Network Functions

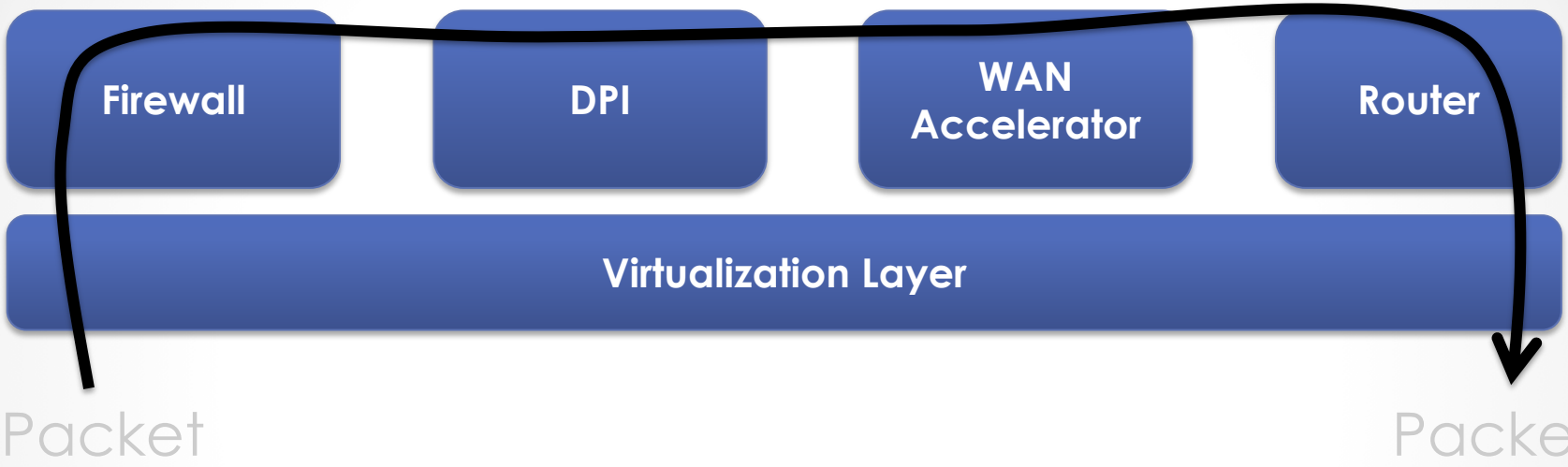


1 Machine : Power x $SW

BUT: Need to provide high bandwidth and low latency

Also: Chained Functionality

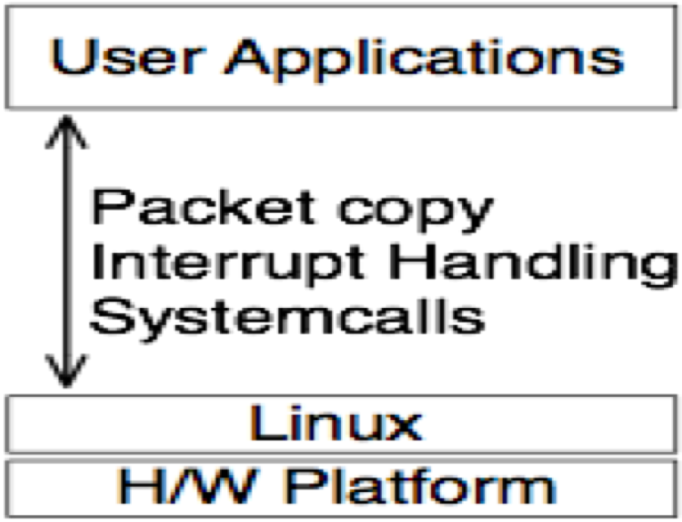
- Functions are often sequential



Need High Speed Inter-Function Communication

Linux Packet Processing

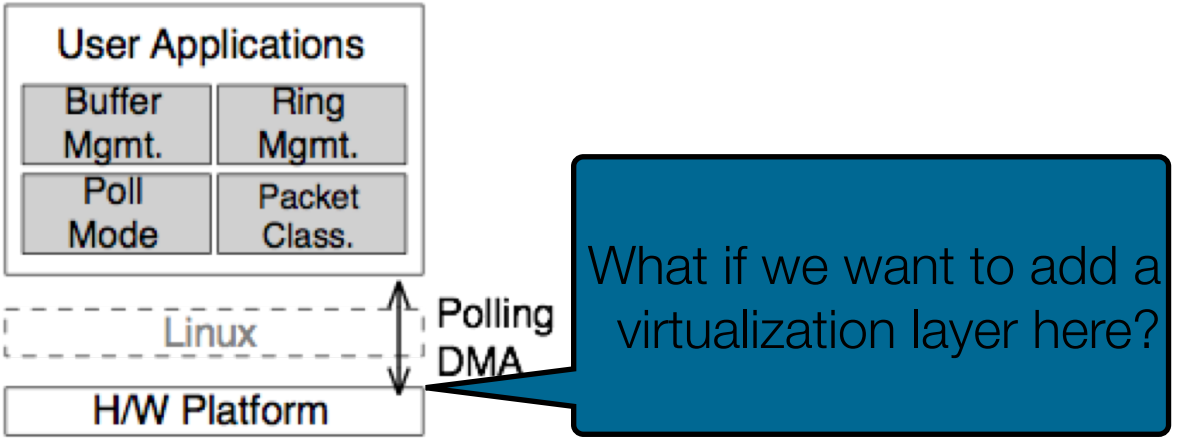
- Traditional networking:
  - NIC uses DMA to copy data into kernel buffer
  - Interrupt when packets arrive
  - Copy packet data from kernel space to user space
  - Use system call to transmit packet from user space



Can it handle being interrupted 14 million times per second?

User Space Packet Processing

- Recent NICs and OS support allows user space apps to directly access packet data
  - NIC uses DMA to copy data into ~~kernel~~ **user space** buffer
  - ~~Interrupt~~ **use polling to find** when packets arrive
  - ~~Copy packet data from kernel space to user space~~
  - Use ~~system~~ **regular function** call to transmit packet from user space



Data Plane Development Kit (DPDK)

Our Contributions

1. **A virtualization-based high-speed packet delivery platform**

   - for flexible network service deployment that can meet the performance of customized hardware, especially those involving complex packet processing

2. **Network shared-memory framework**

   - that truly exploits the DPDK (data plane development kit) library to provide zero-copy delivery to VMs and between VMs

3. **A hypervisor-based switching algorithm**

   - that can dynamically adjust a flow's destination in a state-dependent and/or data-dependent manner

4. **High speed inter-VM communication**

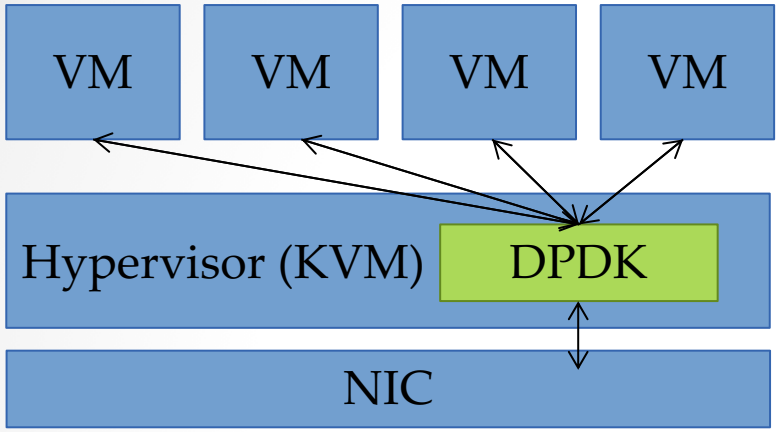
   - enabling complex network services to be spread across multiple VMs

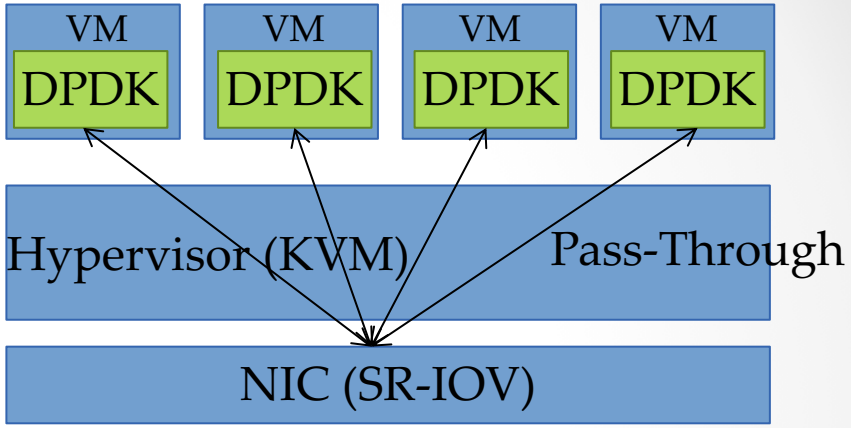
5. **Security domains**

   - that restrict access of packet data to only trusted VMs

DPDK with Virtualization

- Two Architectural Variations





SR-IOV = Single Root I/O Virtualization = Virtual Function Driver Integration

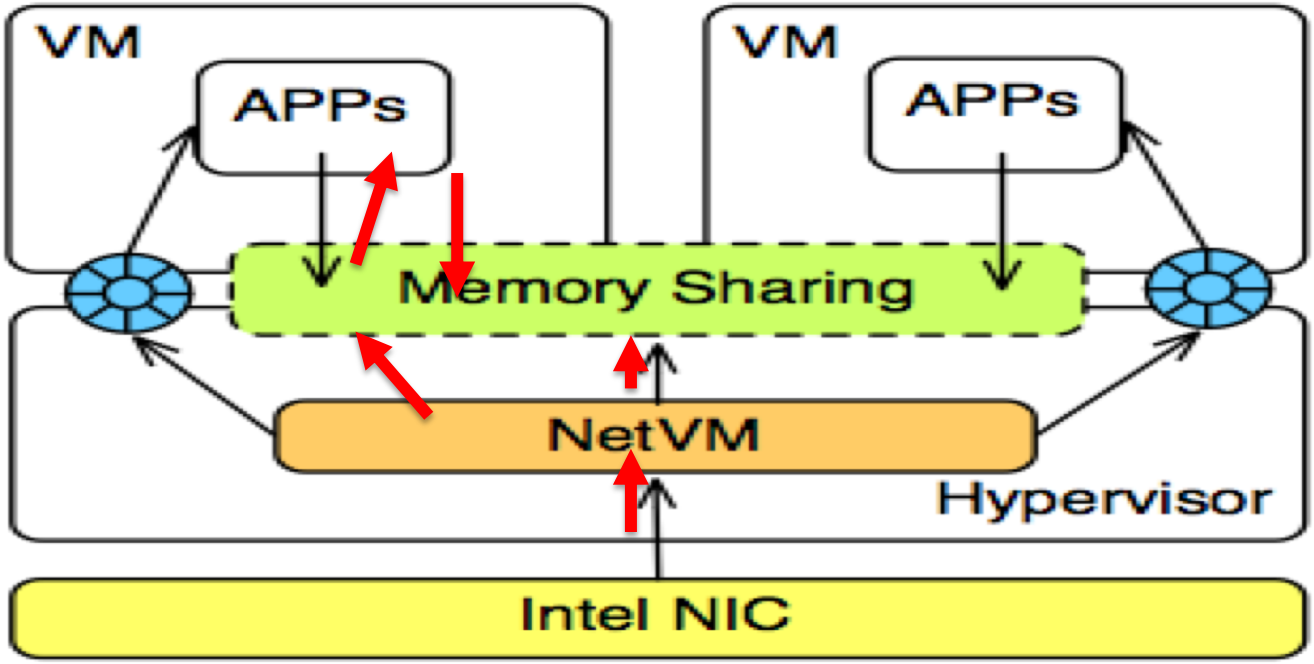
---

- Flexible(dynamic) Configuration
- Control over packet switching
- Control over load-balancing
- Has more overhead

- Max 63 VMs with 2 ports (tx/rx)
- Static configuration
- Inter-VM switch is limited per port
- No control over packet switching
- No control over load-balancing

Both **unable** to achieve full line-rate network b/w in VMs

NetVM Architecture



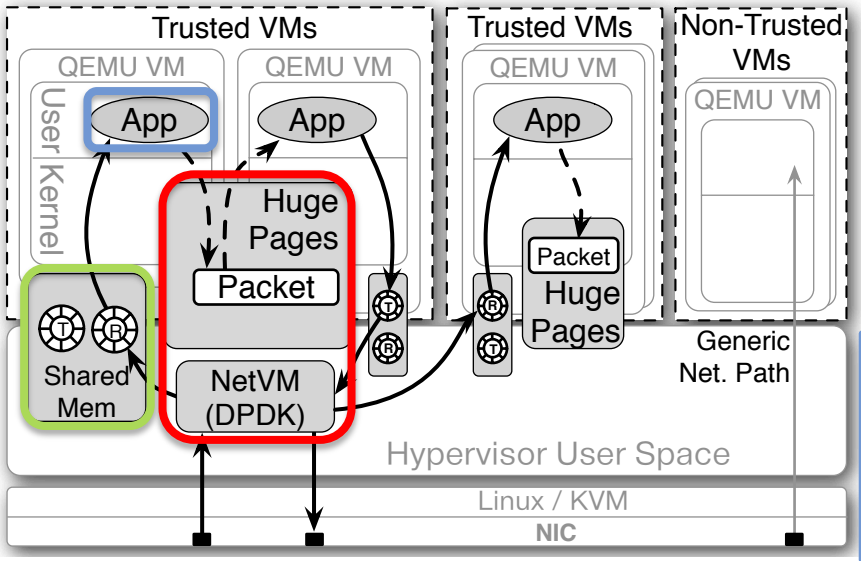
- NetVM (with DPDK) runs in host's User Space
- Memory is shared for network data - zero packet copy
- Each VM has its own ring to receive/transmit packets through descriptors
- Read "NetVM == NF Manager" in this & following slides.

NetVM Architecture

4. NetVM sends a packet in shared memory (zero-copy) via descriptor to VM

5. NetLib reads the packet descriptor and finds the packet in huge page

3. NetVM (w/ DPDK) polls a packet (in huge page) and decides where to send it



6. Application processes the packet and decides what to do next (decision)

2. Run VM & app. in VM, appropriately configured (# cores, callback)

7.NetLib sends packet in shared memory to NetVM via descriptor (zero-copy)

1. Run NetVM application - appropriately configured (NIC, ports, #cores)

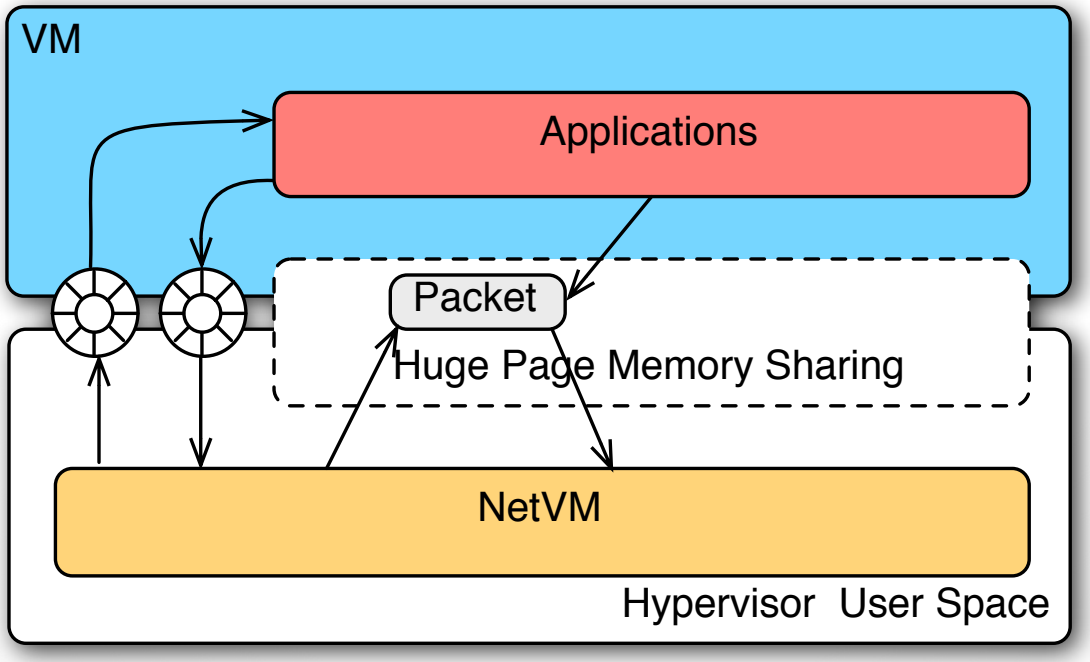
Prerequisites

- Security Domains are pre-defined
- Huge pages are set up
- VMs are not running

8. NetVM receives and transmits, or forwards the packet

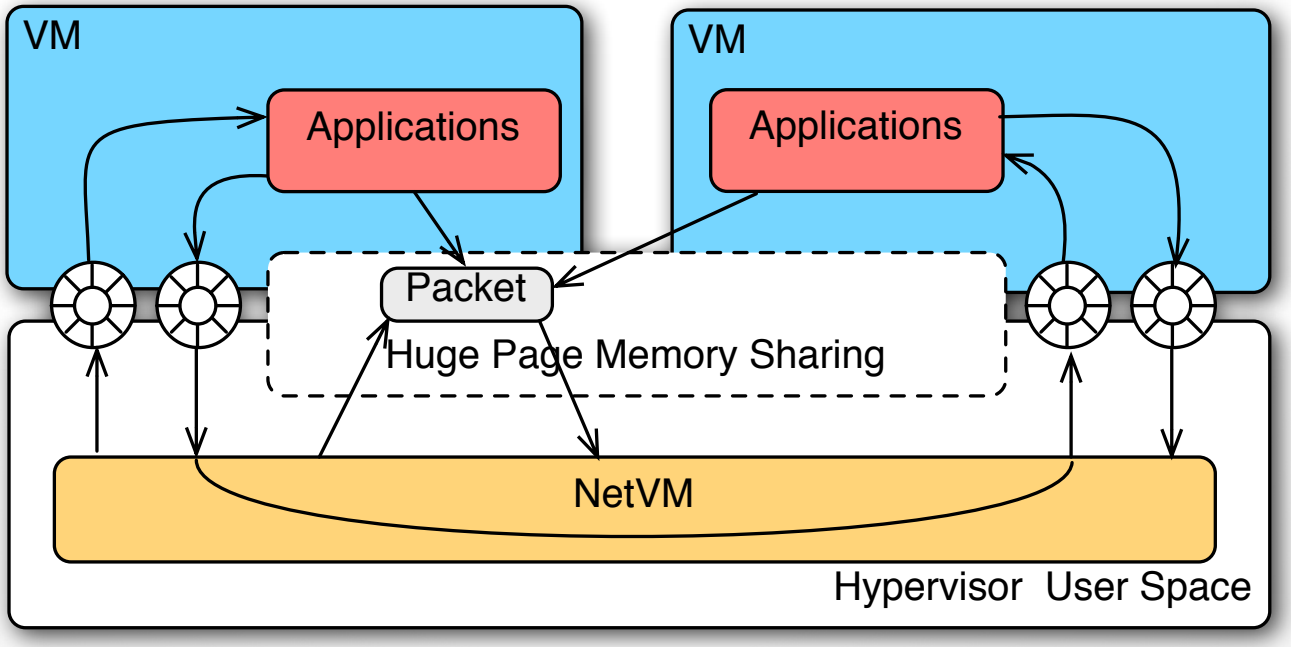
Zero-Copy Packet Delivery

- Packet directly DMA-ed into huge page memory by NIC
- Applications in VM receive references (location) via the shared descriptor ring buffer
- Packet content can be modified by NF application



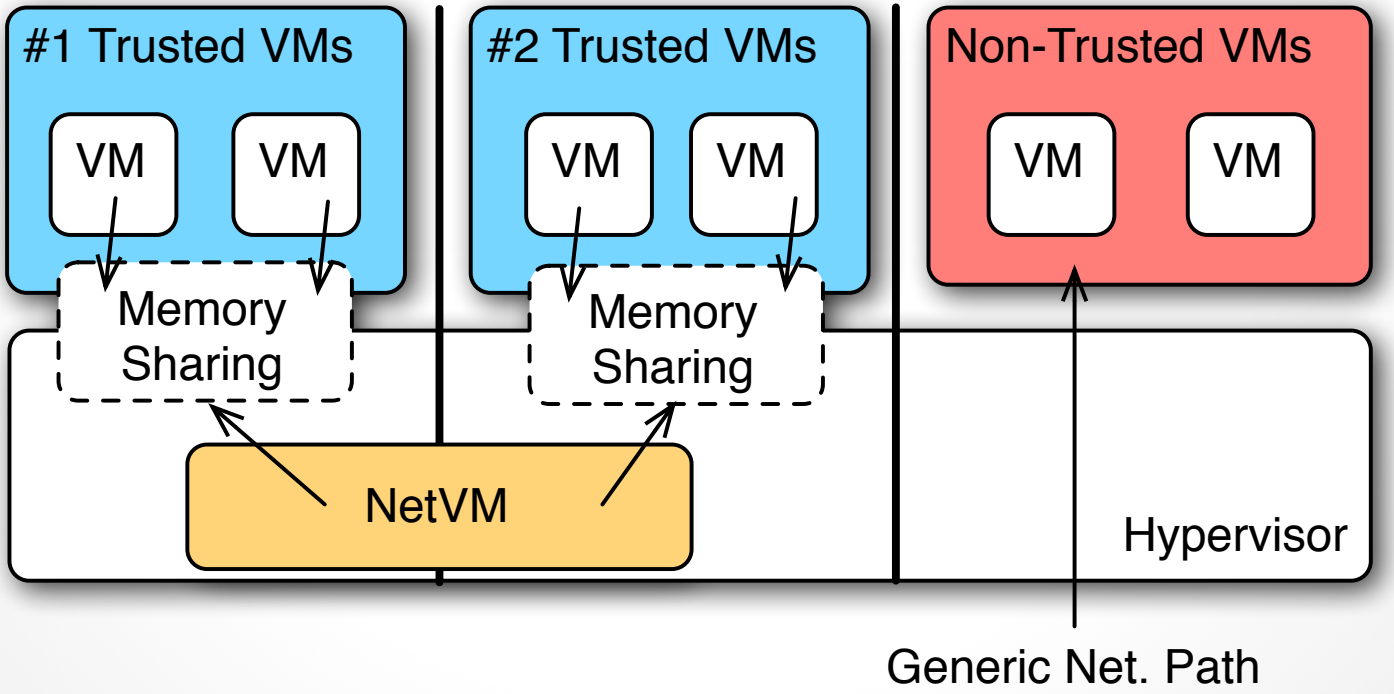
Chained Packet Delivery

- Packets in memory do not have to be copied
- Applications in VMs pass packet references to other VMs – through the descriptor ring
- Only one application can access a given packet at any time (read/write) – avoid locks



Trusted and Untrusted Domains

- Virtualization should provide security guarantees among VMs
- NetVM provides a security boundary between trusted and untrusted VMs
- Untrusted VMs cannot see packets from NetVM
- Grouping of trusted VMs via huge page separation



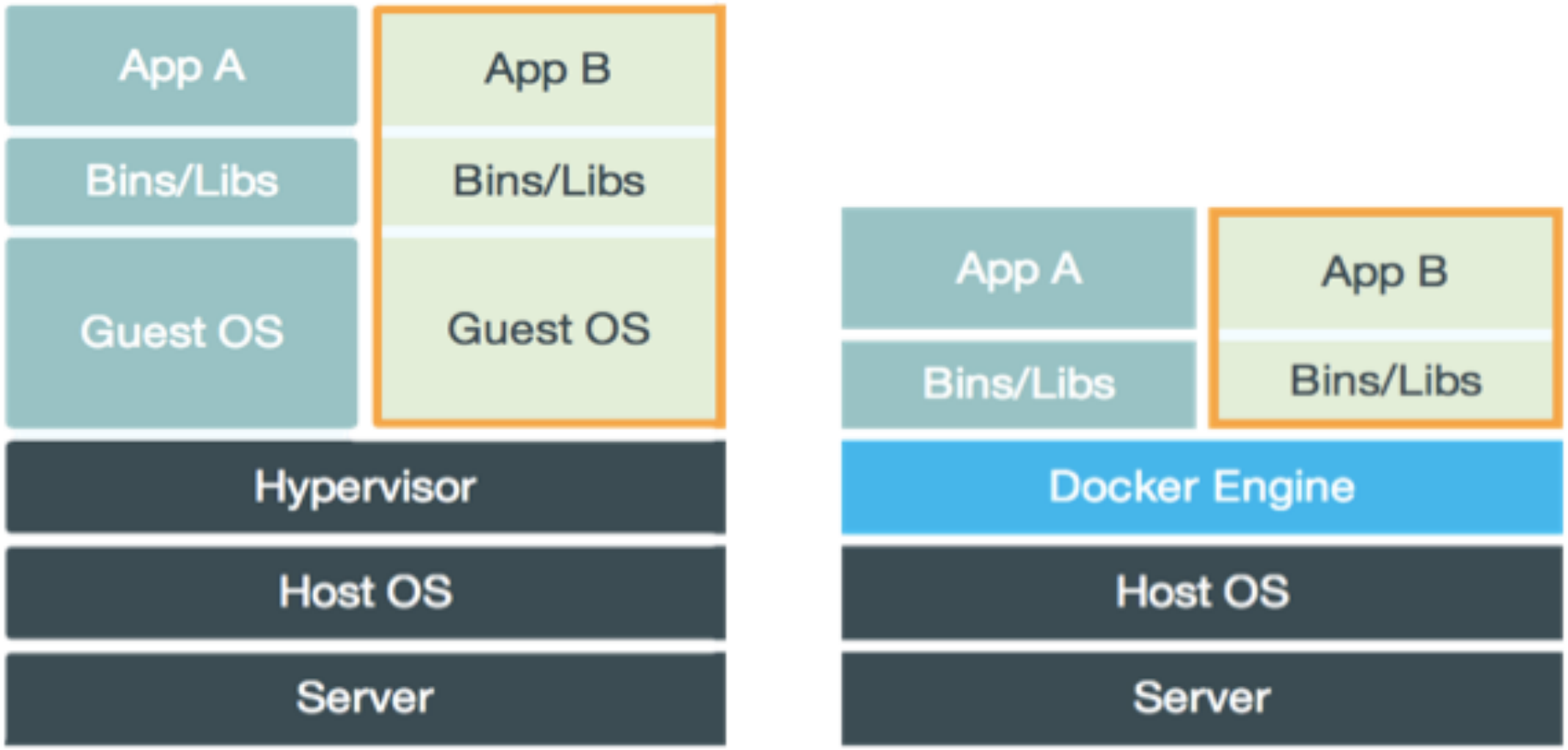
OpenNetVM – Open Source Platform for NFV

- Created an open source version of NetVM
- Moved from KVM Hypervisor and VMs to Docker containers
- NFs run inside Docker containers
- Continue to retain the DPDK based design, to achieve zero-copy high-speed I/O
  - Shared memory across NFs and NF Manager
- Multiple industrial partners evaluating use of OpenNetVM
  - Of course, there are many competitors (e.g., Fast Data Project (fd.io), etc.)

Docker and Containers

- Docker is light-weight and portable
  - Minimal overhead (*cpu/io/network*)
  - Based on Linux containers
  - Can run on any Linux system that supports LXC.
- Self-sufficient
  - A Docker container contains everything it needs to run
  - Minimal Base OS
  - Libraries and frameworks
- A Docker container should be able to run anywhere that Docker can run.

VMs vs. Containers



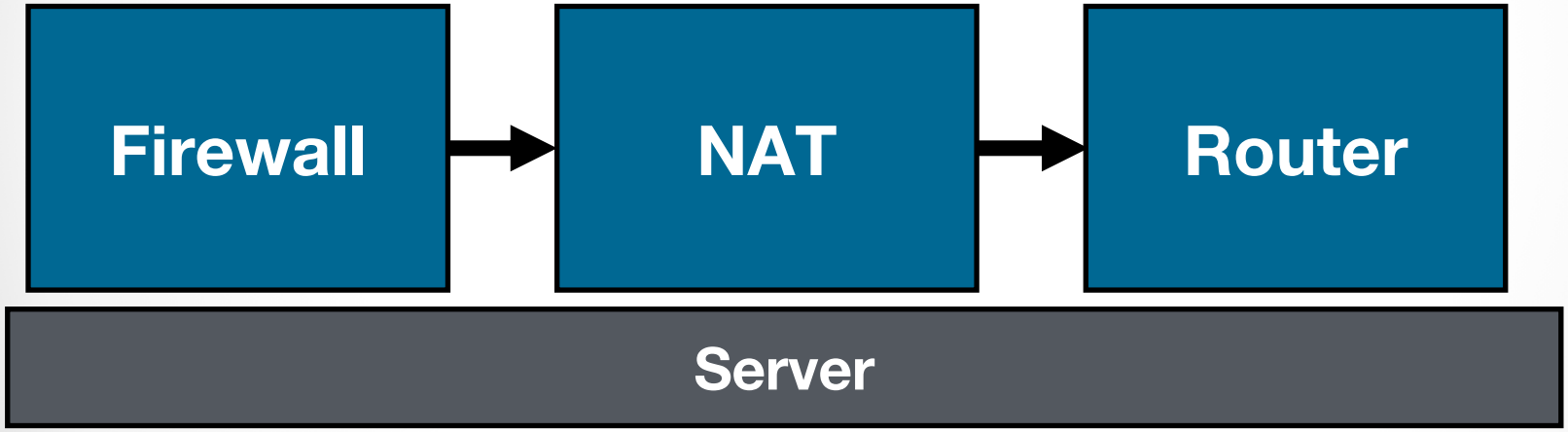
Easy to build, run & share containers

Better performance vs. VMs

Reduces complexity of dynamically instantiating NFs

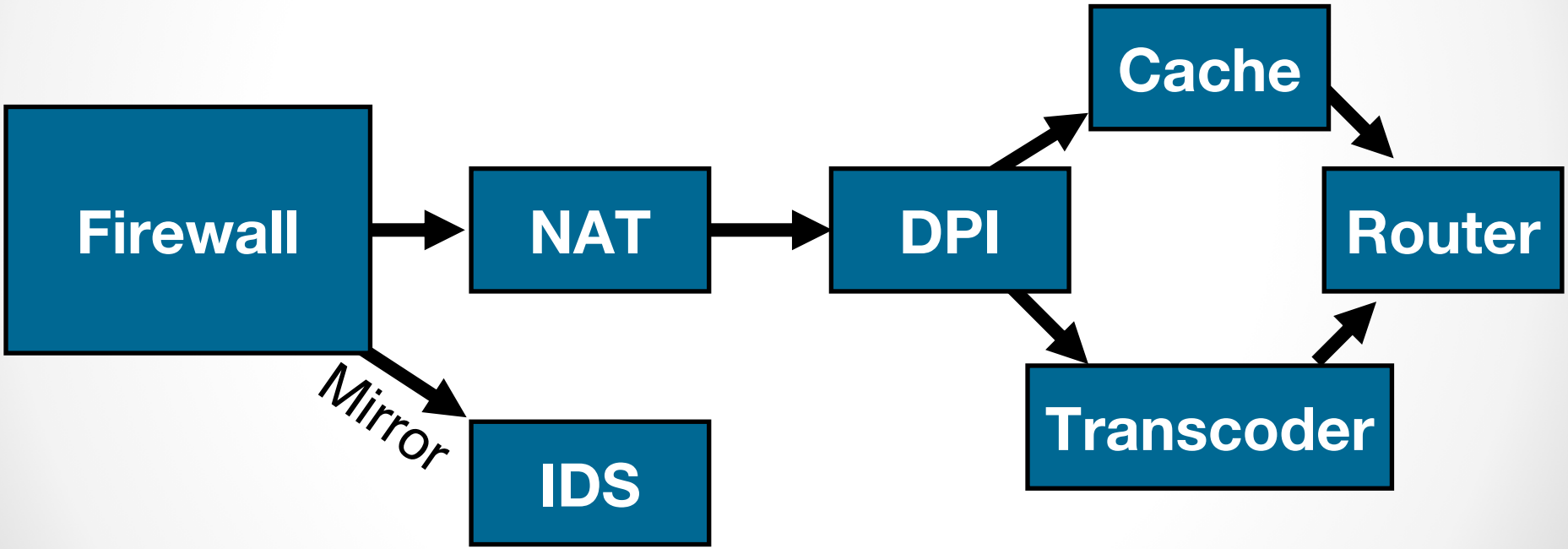
Service Chains

- Chain together functionality to build more complex services
  - Need to move packets through chain efficiently



Service Chains

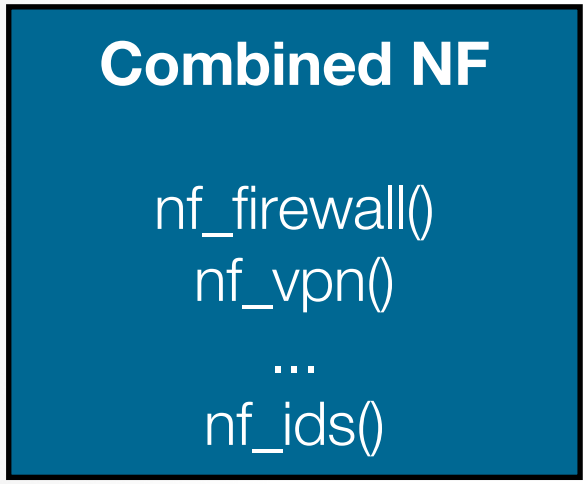
- Chain together functionality to build more complex services
  - Need to move packets through chain efficiently



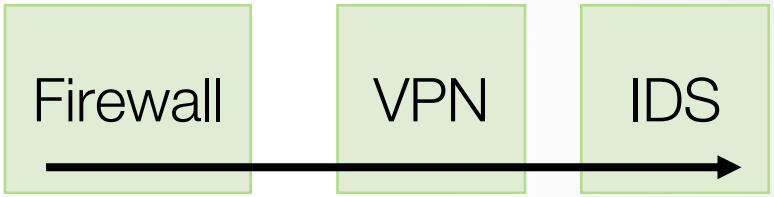
- **Can be complex with multiple paths!**

Design Principle 1

- An NFV platform should focus on providing efficient IO for **modular** NFs which can be **dynamically** deployed and managed



vs



Fast calls to NFs

NFs must be compiled together and known in advance

Tight code dependencies

VM or container per NF provides isolation

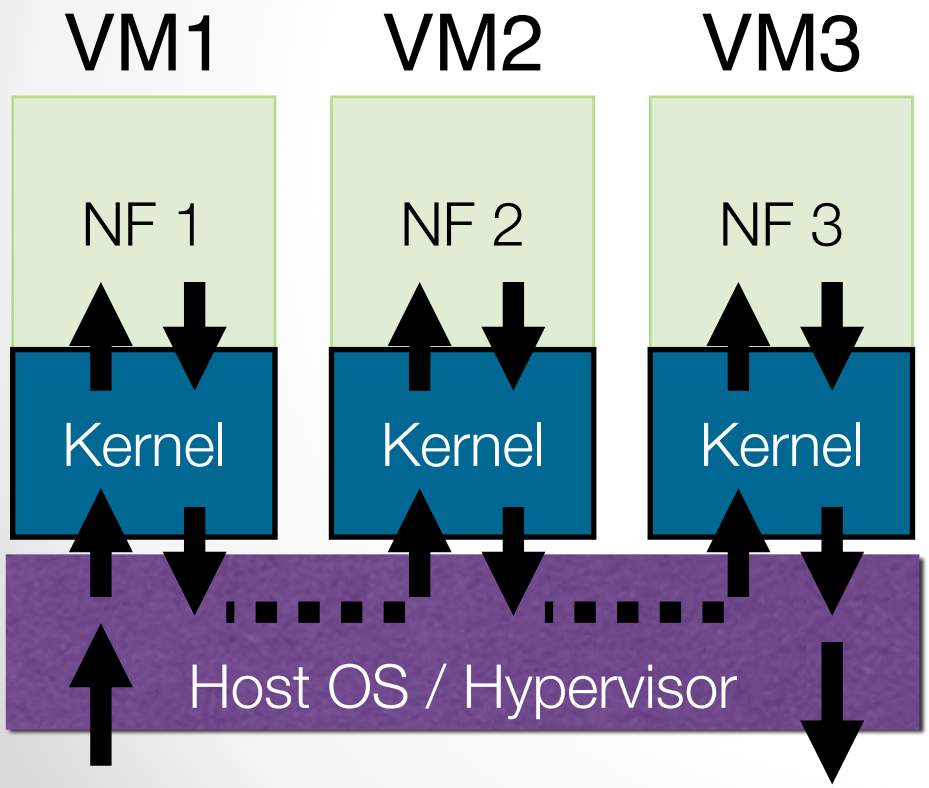
Can be dynamically instantiated

Greater control over resource allocation

Higher cost when moving data between NFs

Packet Copies

- OpenNetVM is designed for NF service chains
  - Don't want to copy packet multiple times to move between NFs
- Existing systems can be very inefficient!

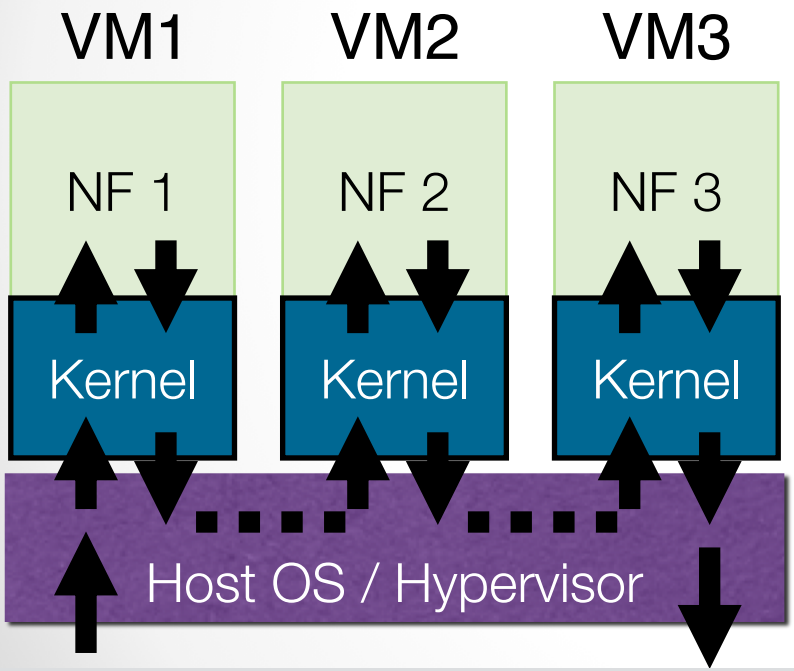


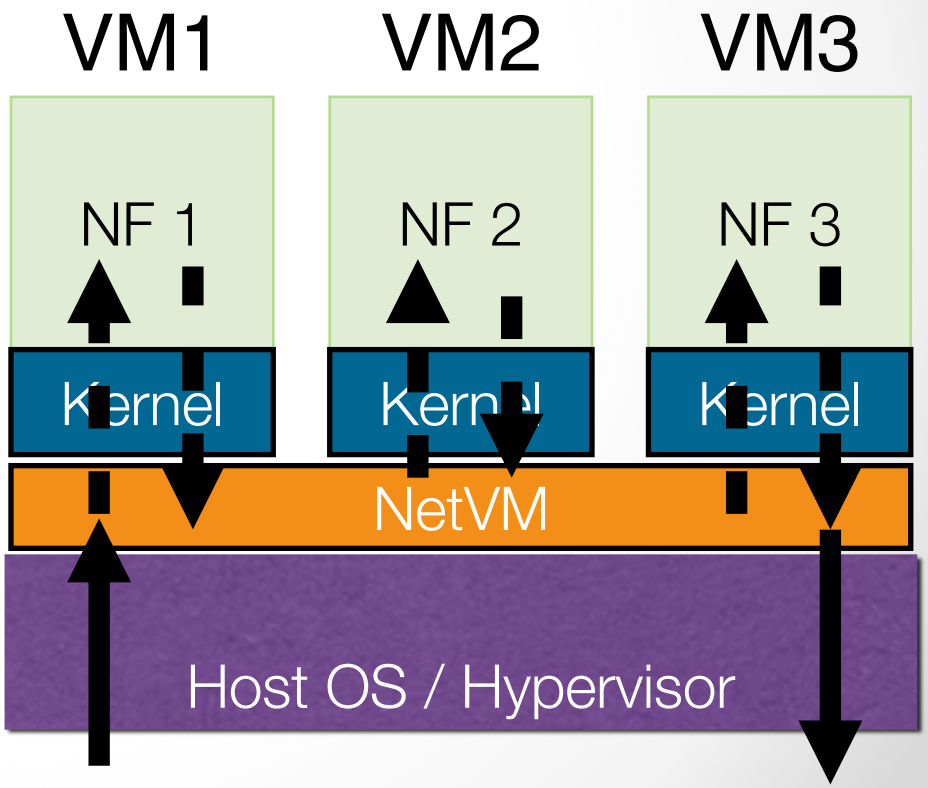
Packet must be copied:
- when going from host to VM
- when going from VM kernel to VM guest

2 DMAs + 12 copies!

Zero Copy I/O

- OpenNetVM is designed for NF service chains
  - Don't want to copy packet multiple times to move between NFs
- DMA into shared memory, directly accessible to NFs!

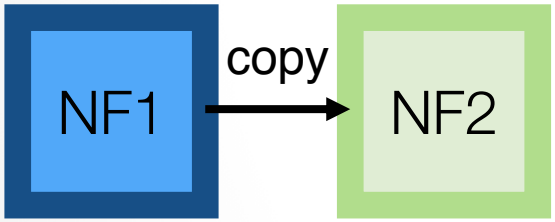




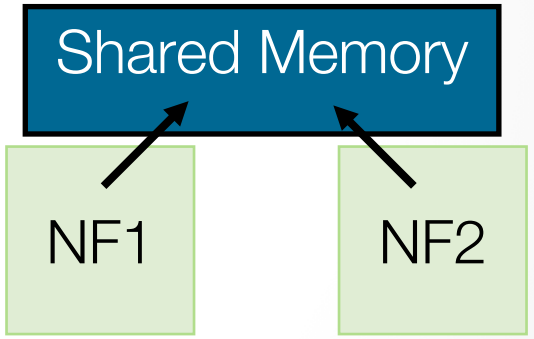
Traditional: 2 DMAs + 12 copies
NetVM: 2 DMAs, 0 copies

Design Principle 2

- Eliminate abstraction layers when they incur overhead or provide protection that is not critical
- NFs are developed by trusted vendors; security is less critical than performance and fault isolation



VS



Protects against packet snooping

High cost in long chains

Eliminates copies while retaining process isolation

Light Weight VMs

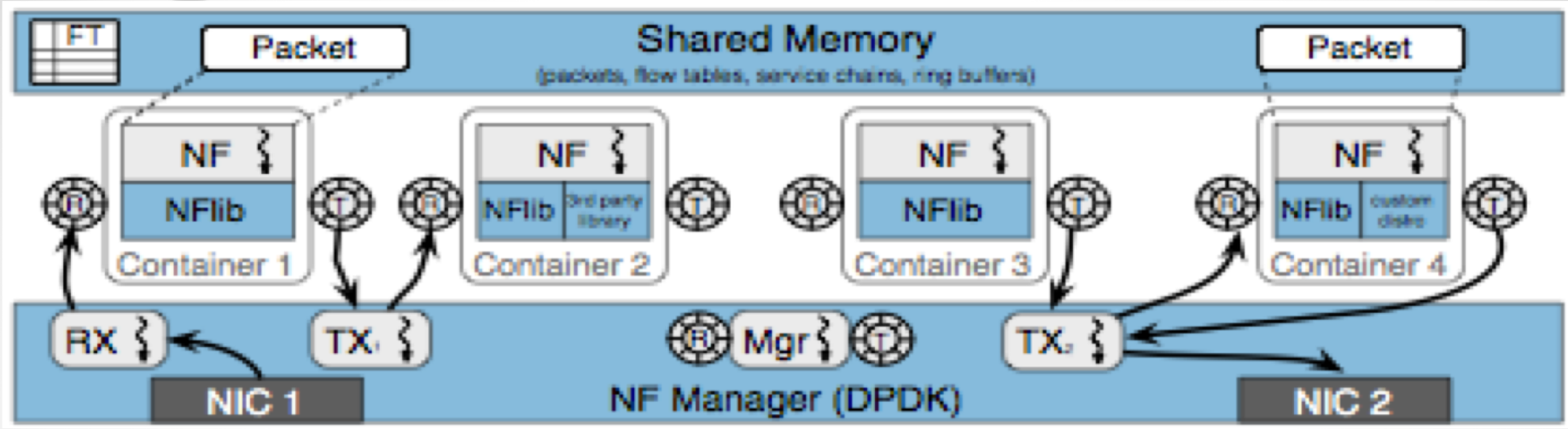
- OpenNetVM uses Docker Containers to run NFs
  - Less overhead than full virtualization
  - Very fast to start new containers (< 0.5 seconds)
  - Easier to share memory
- Much faster service chains compared to ClickOS
  - Negligible overhead from Docker containers

Chain length:	1	2	3	6
ONVM-proc	19.1 Mpps	18.7 Mpps	18.6 Mpps	18.3 Mpps
ONVM-Docker	18.6 Mpps	18.8 Mpps	18.3 Mpps	18.4 Mpps
ClickOS	6.5 Mpps	4.5 Mpps	3.9 Mpps	---

OpenNetVM: NF Containers

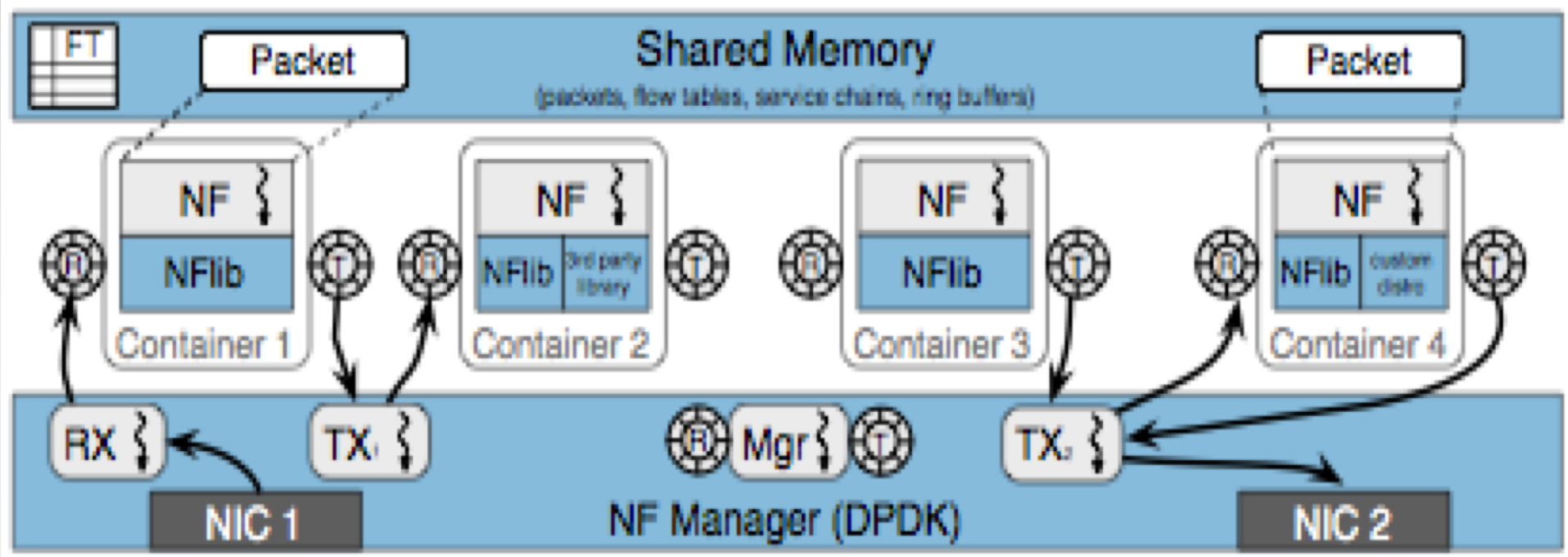
- OpenNetVM encapsulates NFs in containers
- Containers contain all relevant code and files needed for the NF in an easy to deploy package
- Different containers can have different 3rd party libraries or even OS configurations
- Resources can be easily managed at the container level, while providing isolation
- Containers can be dynamically started, stopped, possibly even migrated, all the while being tracked by the OpenNetVM manager
- The OpenNetVM platform enables easy, flexible, BUT HIGH PERFORMANCE communication between the NFs when they are chained together, even if they are produced by distinct vendors.

OpenNetVM Architecture



- **DPDK**: provides underlying I/O engine
- **NFs**: run inside Docker container, use NFlib API
- **Service chains**: connect multiple NFs together
- **NF Manager**: tracks which NFs are active, organizes chains
- **Shared memory**: allows zero-copy packet transfer in chains
  - See [NSDI '14] and [HotMiddlebox '16] for full details

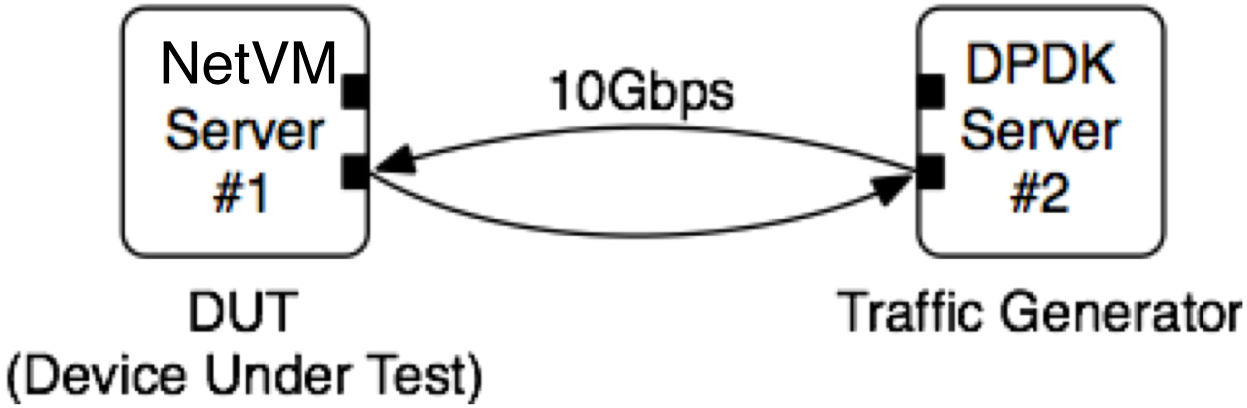
OpenNetVM – System Architecture



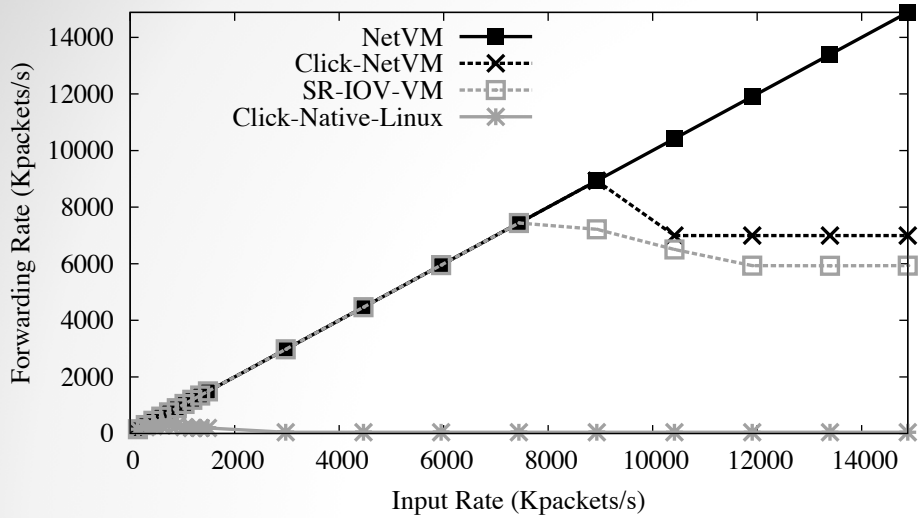
- NF Manager uses DPDK library to allocate memory pools in huge pages for packets
- NFs map memory regions using same base virtual address as NF manager- descriptor access and shared memory
- NFs start in 0.5 seconds; throughput of 68 Gbps w/ 6 cores

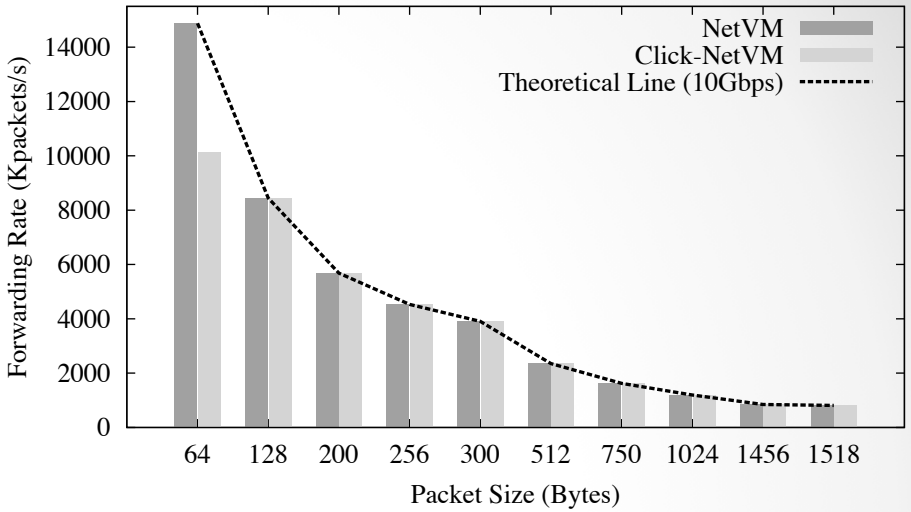
Evaluation Settings

- 2 x Intel-architecture machines
- 82599 Intel NIC
- 2 processors * 6 Cores (no hyperthreading)
  - ✓ 2 cores for receiving packets
  - ✓ 4 cores for transmitting/forwarding packets
  - ✓ Rest of the cores for VM
- Total 8GB Hugepages (4GB for each processor)
- Only 1 port(out of 2 ports) is used
- Apps: L2/L3 Forwarder, Click Userspace Router, Firewall
- Packet Generator: DPDK-Pktgen (Wind River Systems, Inc.)



High Speed Packet Delivery





Left

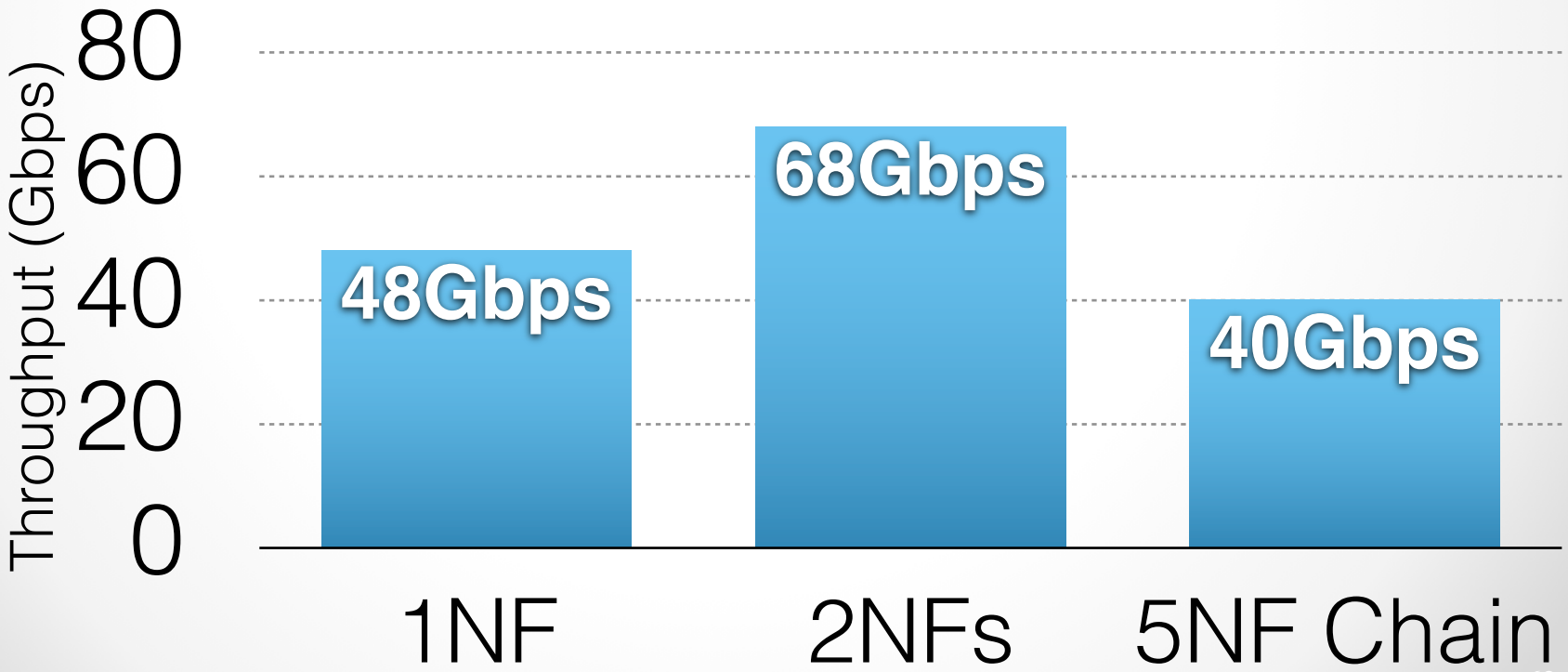
- 64-byte packets, 10Gbps = 14,880,952 packets/s
- NetVM achieves 10Gbps – easily
- Click-NetVM has a maximum of 6Gbps
- SR-IOV achieves a maximum throughput of 5Gbps
- Native Linux system performance is extremely poor

Right

- Forwarding rate when changing packet sizes
- NetVM achieves a full-line rate regardless of packet sizes

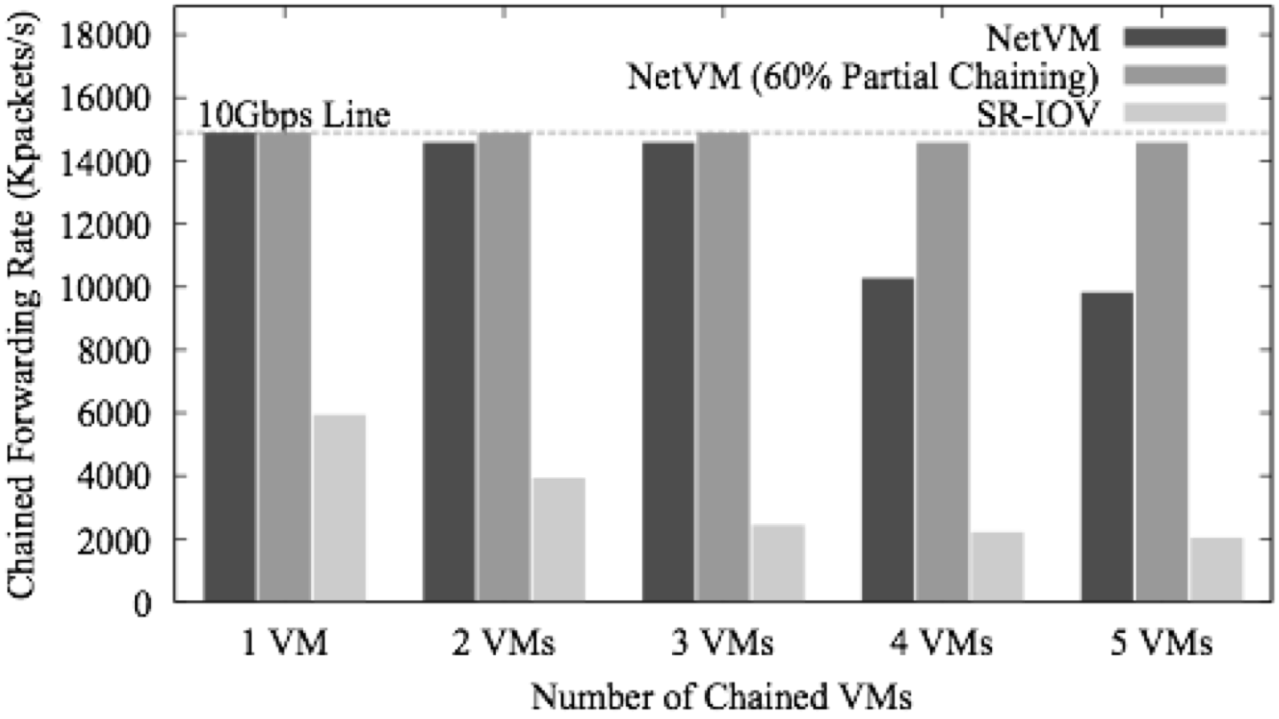
Real Traffic

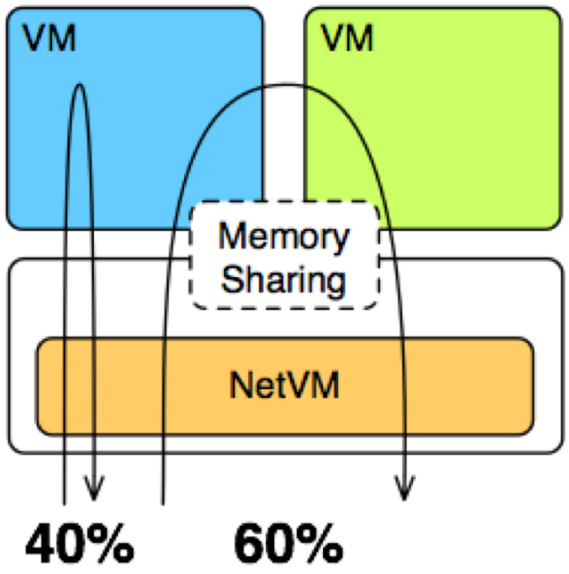
- Send HTTP traffic through OpenNetVM
  - 1 RX thread, 1 TX thread, 1 NF = 50Gbps
  - 2 RX thread, 2 TX thread, 1 NF = 68Gbps (NIC bottleneck)
  - 2 RX threads, 5 TX threads, chain of 5 NFs = 38Gbps
- Fast enough to run a software-based provider edge router



Inter-VM Forwarding

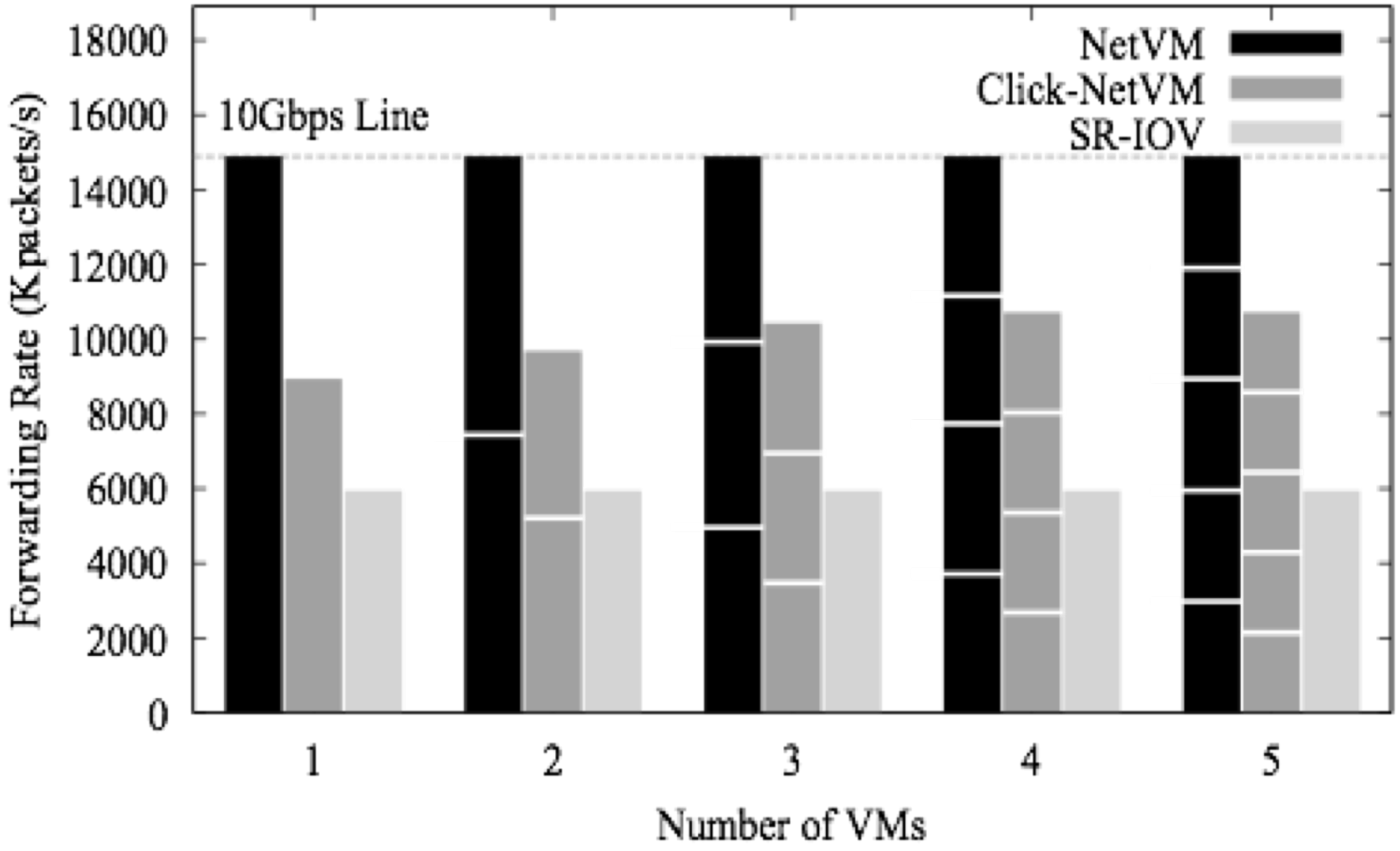
- A mix of L2/L3 forwarding and custom firewall (address filtering)





Switching Flexibility

- State-based load balancing (load at queue)



Getting OpenNetVM

- Source code and NSF CloudLab images at

  http://sdnfv.github.io/