Measuring a 25 Gb/s and 40 Gb/s data plane

Christo Kleu

Pervaze Akhtar

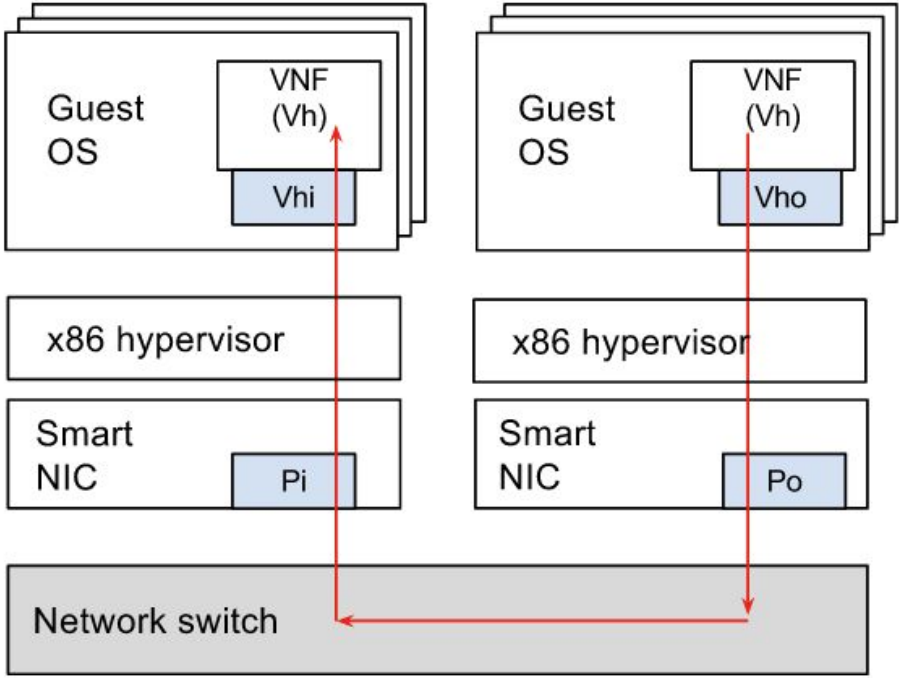
Contents

- **Preliminaries**
  - Equipment
  - Traffic generators
- Test topologies
- Host and VM configuration
  - NUMA Architecture
  - CPU allocation
  - BIOS
  - Host boot settings
  - Interrupt management
- Examples
- Conclusion

Testing Objectives

Testing objective guides choices. Possible objectives:

- Determine throughput limits of SmartNIC processor
- Measure performance of a VNF
  - product VNF
  - experimental VNF
- Packet size:
  - Small packet performance
  - Large packets
  - Packet mix (IMIX)
- Assess:
  - Best case performance
  - Expected performance in a real deployment
- Measure:
  - Packets per second
  - Throughput
  - Packet drop rate? None, limited, unlimited
  - Latency
  - Jitter
- Minimal equipment cost or limited availability



No “one size” solution to everyone’s needs.

Preliminaries - Equipment

- SmartNIC installed in a system.
- Goal is to measure FAST dataplane on the SmartNIC
- System must provide:
    - ARI / SR-IOV
    - Large system RAM. Scale with no. of VMs and VNFs
    - Large number of CPU cores.
    - High clock rate CPUs
- Goal is for the dataplane to be the bottleneck, not host hardware, OS or VNFs

Preliminaries - Traffic generators

- Hardware
  - Ixia
  - Spirent
  - Other high performance hardware generators
- Software
  - DPDK based. High performance. Require sufficient memory to run.
    - Moongen
    - DPDK-pktgen
  - Standard Linux. Simple, lower performance. Must run multiple instances with careful OS and system configuration for valid results
    - Iperf or Iperf3
    - Netperf

Traffic generators - Speed

- Ixia
  - 1G to 100G, any packet size
  - Throughput, latency, jitter, controlled drops
  - Easy to use
- DPDK based generators
  - 15 Mpps per CPU core. (7.5 Gbps of 64 byte packets => 6 CPUs for 40G)
  - Good for 1-10G small packets. We use Ixia for 25G and up.
  - Measures PPS / Gbps. Latency supported with h/w assist from suitable NICs
  - LUA Scriptable
  - Harder to install, medium configuration complexity for high rates
- Iperf / Netperf
  - Throughput (Gbps).
  - Multi-threaded or multi-process operation (16-32 threads or processes, non-linear rate)
  - 1 CPU per thread or process
  - Good for 1-10 Gbps small packets
  - Easy install, more complex configuration for higher rates
  - Affected by OS and kernel version

Contents

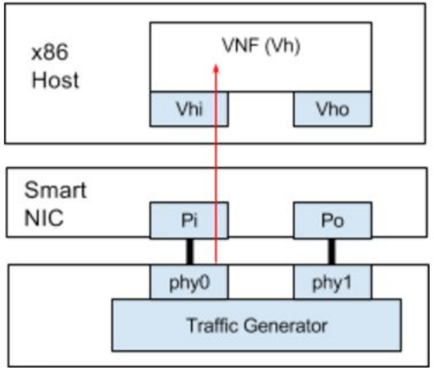
- Preliminaries
    - Equipment
    - Traffic generators
- **Test topologies**
- Host and VM configuration
    - NUMA Architecture
    - CPU allocation
    - BIOS
    - Host boot settings
    - Interrupt management
- Examples
- Conclusion

Test topologies

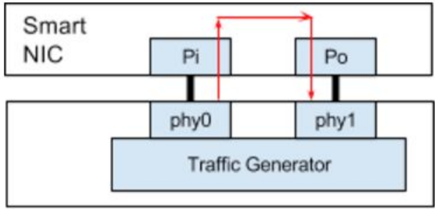
- Single DUT (Device Under Test)
  - Port-to-Host-to-Port
  - Port-to-VM-to-Port
  - Host-to-Host
  - VM-to-VM
  - Port-to-Port (*)
  - Port-to-Host (**)
  - Host-to-Port (**)
- Dual DUT (needed for tunneling)
  - Host-to-Host
  - VM-to-VM

* benchmarking of SmartNIC datapath
** benchmarking of unidirectional datapath

Port-to-Host

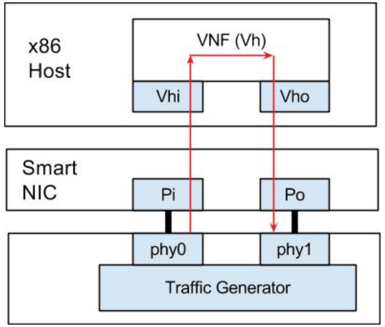


Port-to-Port

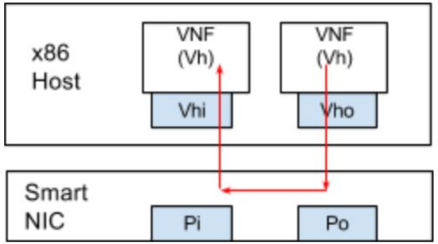


Test topologies - Single DUT

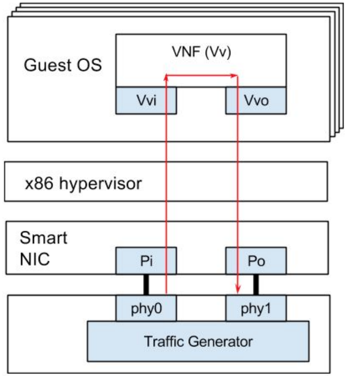
Port-to-Host-to-Port



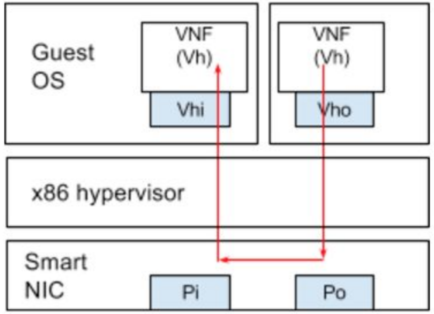
Host-to-Host



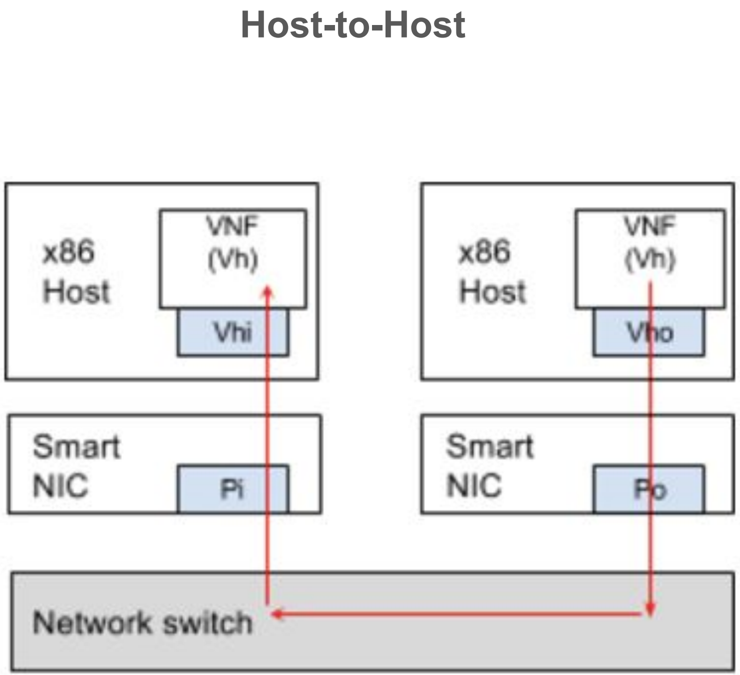
Port-to-VM-to-Port

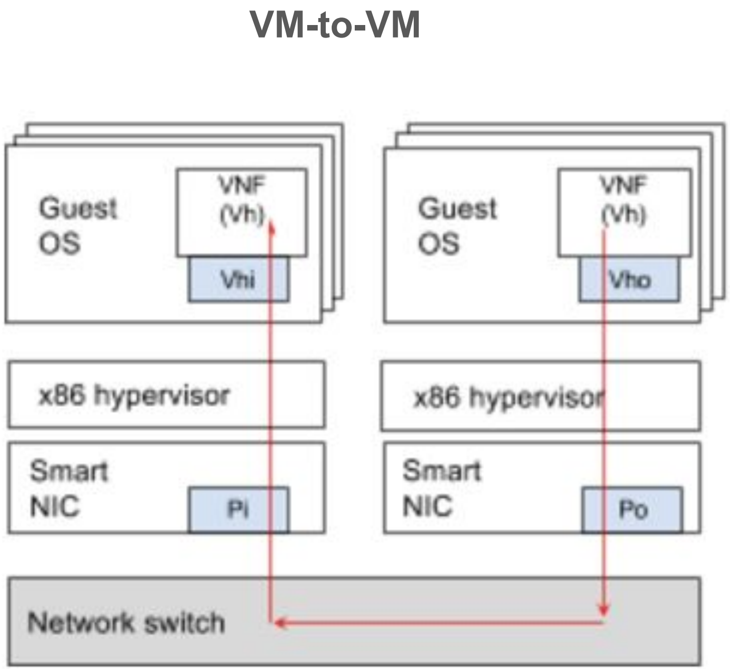


VM-to-VM



Test topologies - Dual DUT





Contents

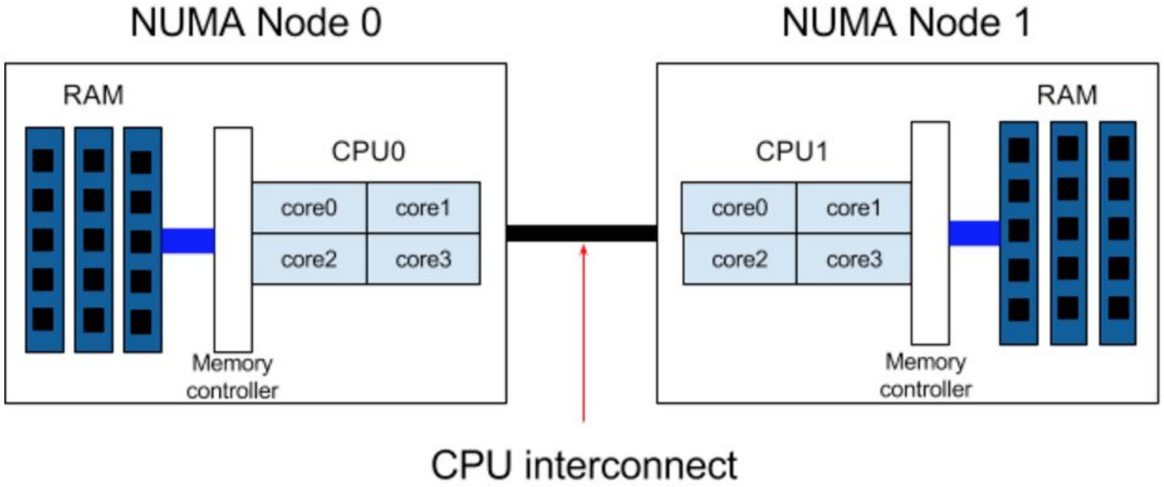
- Preliminaries
  - Equipment
  - Traffic generators
- Test topologies
- **Host and VM configuration**
  - NUMA Architecture
  - CPU allocation
  - BIOS
  - Host boot settings
  - Interrupt management
- Examples
- Conclusion

Configuration goals

- Isolate functional components. Eliminate peturbations
- Use “local” memory
- Eliminate system power management transitions
  - No changes in speed
- Avoid context switches and interruptions in processing
- Minimize handoffs
  - Movements in memory
  - Handoffs from one CPU to another

NUMA Architecture - Introduction

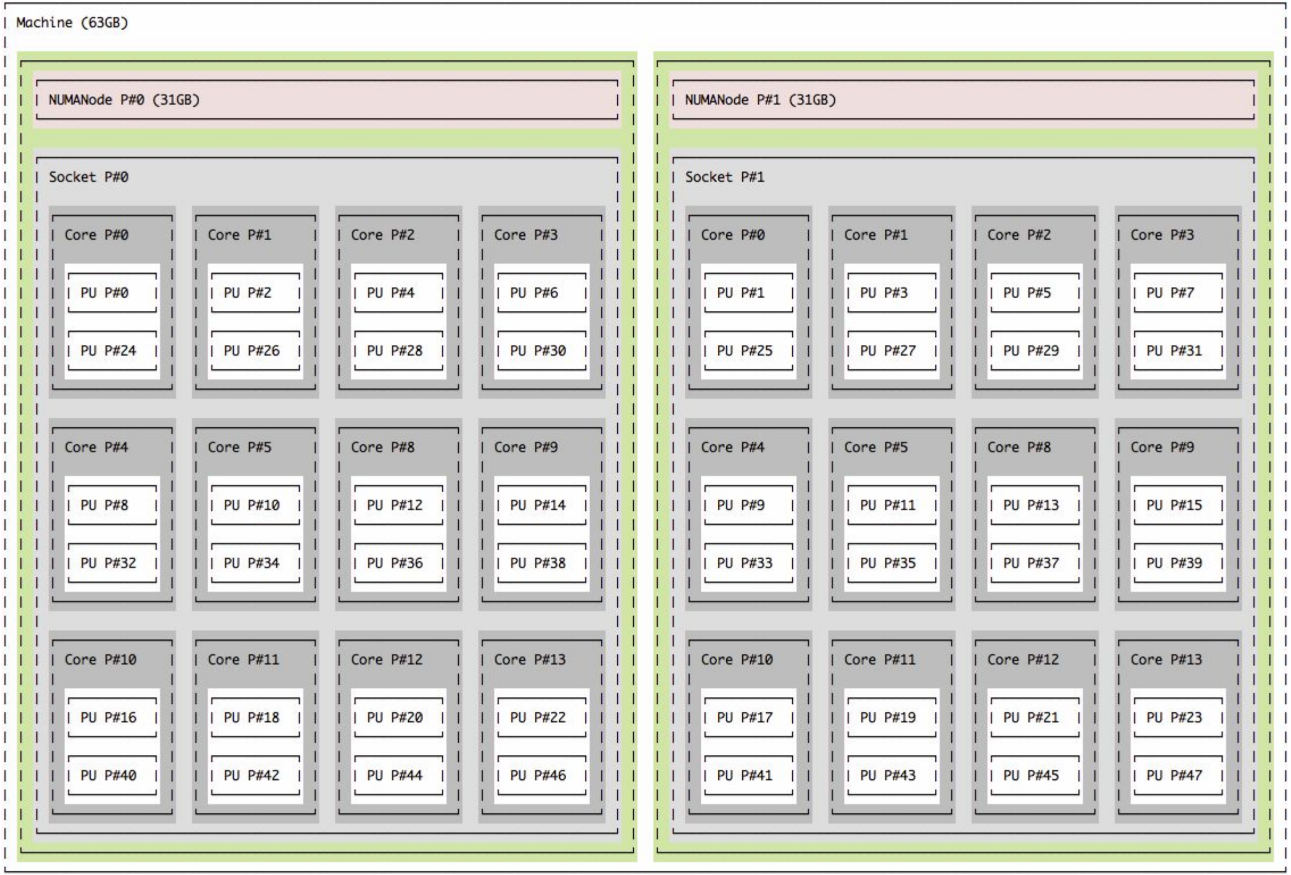
- NUMA topology
- CPU Hyperthread partners
- Memory can be local or remote
- Remote memory is more costly
  - increased latency
  - decreased throughput
- Improve performance:
  - ensure applications are run on specific sets of cores
- Determine the SmartNIC NUMA affinity (used for planning)



NUMA Architecture - Example 1

- NUMA nodes: 2
- Sockets: 2
- Cores per socket: 14
- Threads per core: 2
- CPUs: 48
- 64GB RAM
- 32GB RAM per NUMA node
- NUMA node0
  - 0,2,4,.. (even)
- NUMA node1
  - 1,3,5,.. (odd)

```
lstopo
lscpu
numactl
```



```
Machine (63GB)

NUMANode P#0 (31GB)                            NUMANode P#1 (31GB)

Socket P#0                                     Socket P#1

Core P#0  Core P#1  Core P#2  Core P#3         Core P#0  Core P#1  Core P#2  Core P#3
PU P#0    PU P#2    PU P#4    PU P#6           PU P#1    PU P#3    PU P#5    PU P#7
PU P#24   PU P#26   PU P#28   PU P#30          PU P#25   PU P#27   PU P#29   PU P#31

Core P#4  Core P#5  Core P#8  Core P#9         Core P#4  Core P#5  Core P#8  Core P#9
PU P#8    PU P#10   PU P#12   PU P#14          PU P#9    PU P#11   PU P#13   PU P#15
PU P#32   PU P#34   PU P#36   PU P#38          PU P#33   PU P#35   PU P#37   PU P#39

Core P#10 Core P#11 Core P#12 Core P#13        Core P#10 Core P#11 Core P#12 Core P#13
PU P#16   PU P#18   PU P#20   PU P#22          PU P#17   PU P#19   PU P#21   PU P#23
PU P#40   PU P#42   PU P#44   PU P#46          PU P#41   PU P#43   PU P#45   PU P#47
```

NUMA Architecture - Example 2

```
compute-02-002:~# lscpu | grep NUMA
NUMA node(s):          2
NUMA node0 CPU(s):     0-9,20-29
NUMA node1 CPU(s):     10-19,30-39

compute-04-003:~# lscpu | grep NUMA
NUMA node(s):          2
NUMA node0 CPU(s):
0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46
NUMA node1 CPU(s):
1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47
```

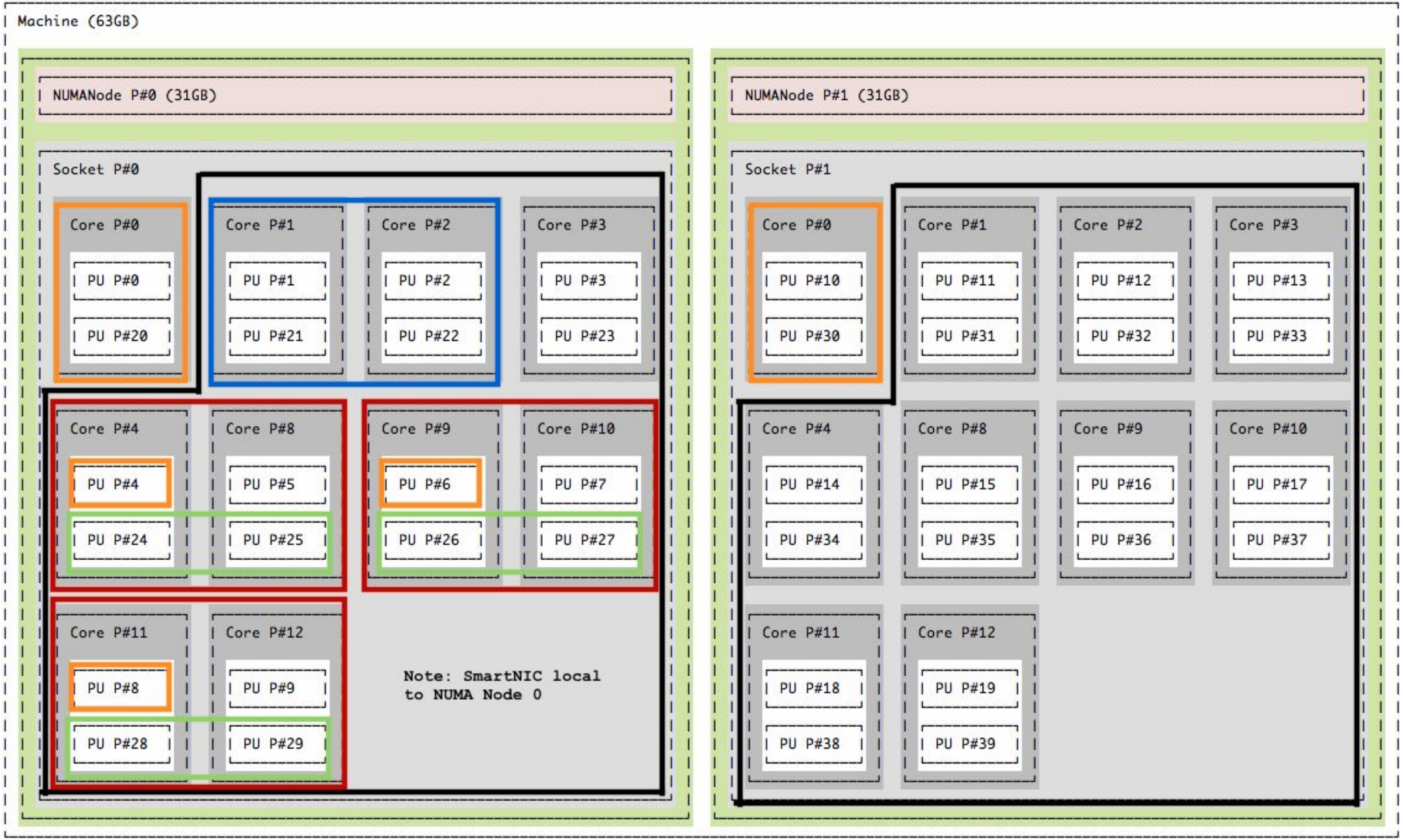
CPU affinity and pinning

- CRITICAL - Do not use the same CPU for more than one task
- Isolate and pin CPUs dedicated for critical tasks
- Pin VMs to unique CPUs
    - `vcpupin <domain> [--vcpu <number>] [--cpulist <string>]`
- Pin DPDK apps to unique CPUs (or logical cores in VM)
    - use `-c <coremap>` or `--l <corelist>` to specify CPUs DPDK app uses

CPU allocation - Example

LEGEND

- Isolated CPUs
- Host/VM OS CPUs
- VirtIO relay CPUs
- VM CPUs
- VM DPDK app CPUs



BIOS settings (1/3)

- **C-state disabled**. Ensure that the CPUs are always in the C0 (fully operational) state. Each CPU has several power modes and they are collectively called “C-states” or “C-modes.”.
- **P-state disabled**. The processor P-state is the capability of running the processor at different voltage and/or frequency levels. Generally, P0 is the highest state resulting in maximum performance, while P1, P2, and so on, will save power but at some penalty to CPU performance.
- **Hyper-threading** enabled/disabled.
- **Turbo Boost Disabled**. Turbo Boost has a series of dynamic frequency scaling technologies enabling the processor to run above its base operating frequency via dynamic control. Although Turbo Boost (overclocking) would increase the performance, not all cores would be running at the turbo frequency and it would impact the consistency of the performance tests, thus it would not be a real representation of the platform performance.
- **System Management Interrupts disabled**. Disabling the Processor Power and Utilization Monitoring SMI has a great effect because it generates a processor interrupt eight times a second in G6 and later servers. Disabling the Memory Pre-Failure Notification SMI has a smaller effect because it generates an interrupt at a lower frequency

BIOS settings (2/3)

- Correct setting of BIOS for DUT (w/ SmartNIC) is important, to address:
  - Throughput variations due to CPU power state changes (C-state; P-state)
  - Blips (short duration packet drops) due to interrupts (SMI)
- Example settings (HP) changes - settings will vary from BIOS to BIOS
- Power Management
  - `Power Regulator  = [Static High Performance]` (was “dynamic power savings”)
  - `Minimum Processor Core Idle C-state  = [No C-states]` (was C-6)
  - `Minimum Package Idle C-state = [ No package state]` (was C-6)
  - `Power Profile = [ Max Performance ]` (was “customer”) this ended up setting the above anyway (and graying them out)
- Prevent CPUs from entering power savings modes as transitions cause OS and VM drivers to drop packets.

BIOS settings (3/3)

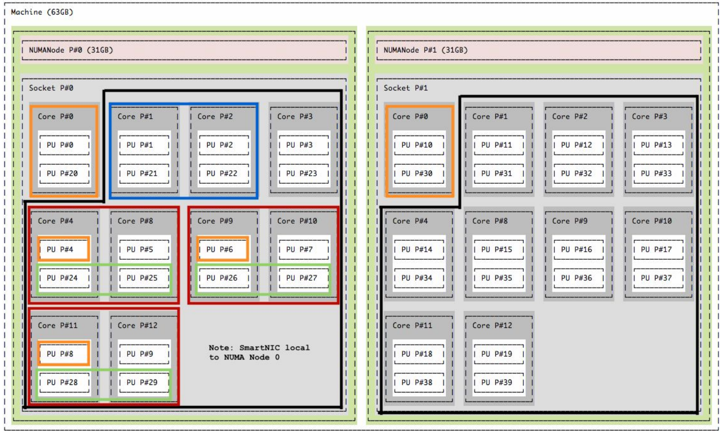
- Example settings (HP) changes (cont.)
- SMI - System Management Interrupts
    - `Memory pre-failure notification = [disabled]` (was enabled)
- When enabled, CPU is periodically interrupted, resulting in packet drops when close to throughput capacity.
    - Disable all non-essential SMIs
    - Frequency of SMI varies
        - On an HP DL380 gen9 it is every 60s
        - On gen6,7 it is every hour
        - On gen8 every 5 minutes

Host boot settings - Power Management

- Make changes carefully!
- Power management
  - `intel_idle.max_cstate=0 processor.max_cstate=0 idle=mwait intel_pstate=disable`
  - `idle=mwait` C1 will be entered whenever the core is idle irrespective of other settings
  - `idle=poll` keeps the processing cores in C0 state when used in conjunction with `intel_idle.max_cstate=0`
- Note: These are in addition to the BIOS settings

Host boot settings - CPU isolation

- CRITICAL - shared CPUs yield poor performance
- The `isolcpus` Linux kernel parameter is used to isolate the cores from the general Linux scheduler
- Plan out CPU allocation, e.g.
  - CPUs should NOT be used for more than one task.
  - CPUs for host OS
  - CPUs for XVIO
  - CPUs for VMs
    - CPUs for guest OS
    - CPUs for VNF or application



Host boot settings - hugepages

- Settings required if hugepages are required by the applications
  - XVIO (Netronome module providing DPDK to virtio acceleration)
  - DPDK applications (e.g. packet generators) in host or VM
- **Hugepages** is a feature that allows the Linux kernel to utilize the multiple page size capabilities of modern hardware architectures.
  - A **page** is the basic unit of virtual memory, with the default page size being 4 KB in the x86 architecture.
- Leave sufficient memory for OS use
  - no longer subject to normal memory allocations
  - Huge Pages are ‘pinned’ to physical RAM and cannot be swapped/paged out.
- Preference is for 1G hugepages.
  - Each hugepage requires a TLB entry. Smaller hugepages => more TLBs and TLB lookups due to page faults => higher probability of packet drop blips

```
default_hugepagesz=1G hugepagesz=1G hugepages=224
```

Interrupt management

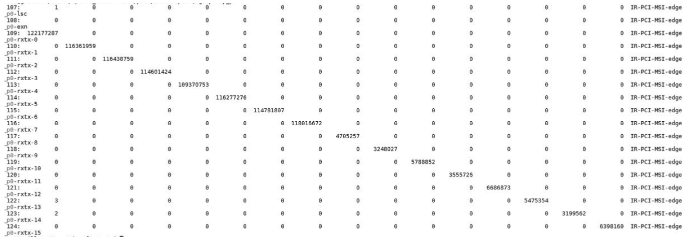
- `irqbalance` keeps all of the IRQ requests from backing up on a single CPU
- manual pinning for performance testing of low latency real-time applications
- disable `irqbalance` (and other affinity management utilities: `numad`, `tuned`)
- It can be beneficial to turn off IRQ balancing on the host.
  - IRQ processing will be limited to non-isolated CPUS

```
service irqbalance stop
/etc/default/irqbalance
```

- `Pin interrupts to non-dataplane CPUs`

Interrupt management - Pin IRQs

- Check the IRQ indexes of the port
  - Does the interface support multiple queues?
  - `/proc/interrupts`
- IRQ affinity (`smp_affinity`) specifies the CPU cores that are allowed to execute the ISR for a specific interrupt.
  - Assign interrupt affinity and the application's thread affinity to the same set of CPU cores to allow cache line sharing between the interrupt and application threads.
  - Equally distribute IRQS between available CPUS



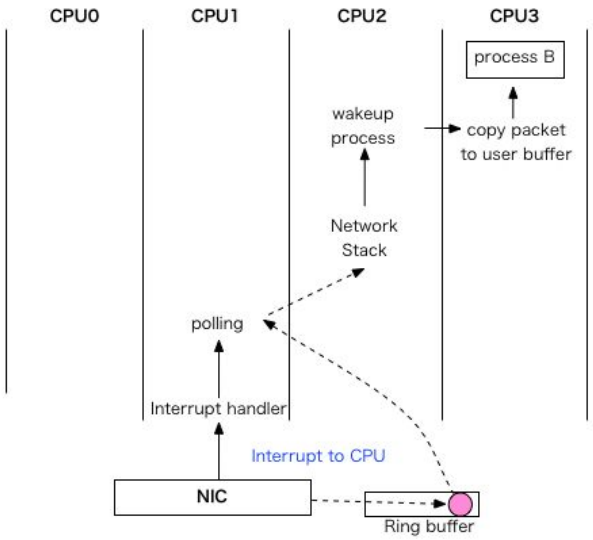
```
 107:          1          0          0          0          0          0          0          0          0          0          0          0          0          0          0          0  IR-PCI-MSI-edge
_p0-lsc
 108:          0          0          0          0          0          0          0          0          0          0          0          0          0          0          0          0  IR-PCI-MSI-edge
_p0-exn
 109:  122177287          0          0          0          0          0          0          0          0          0          0          0          0          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-0
 110:          0  116361959          0          0          0          0          0          0          0          0          0          0          0          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-1
 111:          0          0  116438759          0          0          0          0          0          0          0          0          0          0          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-2
 112:          0          0          0  114601424          0          0          0          0          0          0          0          0          0          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-3
 113:          0          0          0          0  109370753          0          0          0          0          0          0          0          0          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-4
 114:          0          0          0          0          0  116277276          0          0          0          0          0          0          0          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-5
 115:          0          0          0          0          0          0  114781807          0          0          0          0          0          0          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-6
 116:          0          0          0          0          0          0          0  118016672          0          0          0          0          0          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-7
 117:          0          0          0          0          0          0          0          0    4705257          0          0          0          0          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-8
 118:          0          0          0          0          0          0          0          0          0    3248027          0          0          0          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-9
 119:          0          0          0          0          0          0          0          0          0          0    5788852          0          0          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-10
 120:          0          0          0          0          0          0          0          0          0          0          0    3555726          0          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-11
 121:          0          0          0          0          0          0          0          0          0          0          0          0    6686873          0          0          0  IR-PCI-MSI-edge
_p0-rxtx-12
 122:          3          0          0          0          0          0          0          0          0          0          0          0          0    5475354          0          0  IR-PCI-MSI-edge
_p0-rxtx-13
 123:          2          0          0          0          0          0          0          0          0          0          0          0          0          0    3199562          0  IR-PCI-MSI-edge
_p0-rxtx-14
 124:          0          0          0          0          0          0          0          0          0          0          0          0          0          0          0    6398160  IR-PCI-MSI-edge
_p0-rxtx-15
```

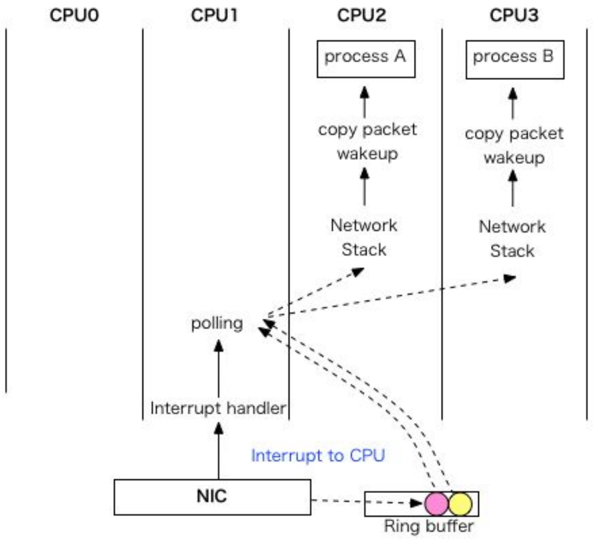
- Pin an IRQ to a core
  - `echo $IRQ_CPU_LIST > /proc/irq/$IRQ/smp_affinity_list`

Netdevs and interrupts

When using netdevs (often done with simple iperf / netperf test configurations)

- Interrupt management is very important
- RPS - receive packet steering (left)
- RFS - receive flow steering (supplements RPS) (right)





Interrupt management - VM settings

- Disable IRQ balancing in VM to prevent data processing CPUs from receiving interrupts.
- Pin apps to vcpus
  - Examples for l2fwd dpdk app:
    - `l2fwd -c 6 -n 4 — -p 3 -T 0` # 3 vCPU VM
    - `l2fwd -c 7 -n 4 — -p 7 -T 0` # 4 vCPU VM
    - Note: “`-c <coremap>`” tells dpdk app which cores to run on.
    - Note: “`-p <PORTMASK>`” tells dpdk app which ports to use.

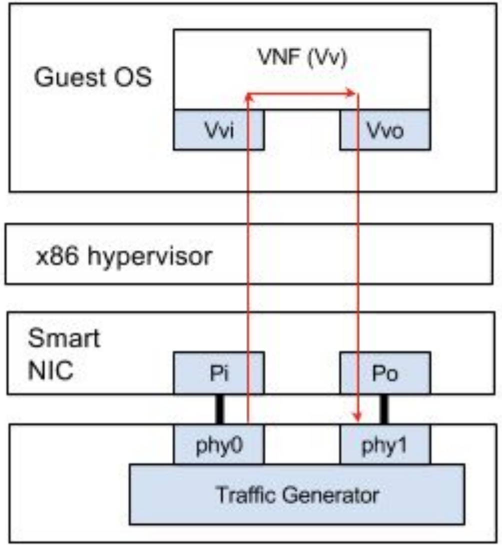
Contents

- Preliminaries
  - Equipment
  - Traffic generators
- Test topologies
- Host and VM configuration
  - NUMA Architecture
  - CPU allocation
  - BIOS
  - Host boot settings
  - Interrupt management
- **Examples**
- Conclusion

Agilio OVS - Example tuning setup (1/5)

Testing goal

- Verify layer 2 forwarding performance of a VNF (entire datapath including virtualization)
- Is the VNF the bottleneck?
- Is the PCI-E bus the bottleneck?
- Is the SmartNIC the bottleneck?
- Is the QSFP the bottleneck?
- Is OVS the bottleneck?



Testing topology

- Host-to-VM-to-Host (Single DUT)
- Traffic generator: Ixia

CPU allocation design

- Host OS
- virtiorelay
- VM00, VM01, VM02, VM03

Agilio OVS - Example tuning setup (2/5)

```
# lspci -d 19ee:4000
05:00.0 Ethernet controller: Netronome Systems, Inc. Device 4000

# cat /sys/bus/pci/drivers/nfp/0000:05:00.0/numa_node
0

# cat /sys/bus/pci/drivers/nfp/0000:05:00.0/local_cpulist
0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46

# lscpu | grep 'NUMA'
NUMA node(s):          2
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47
```

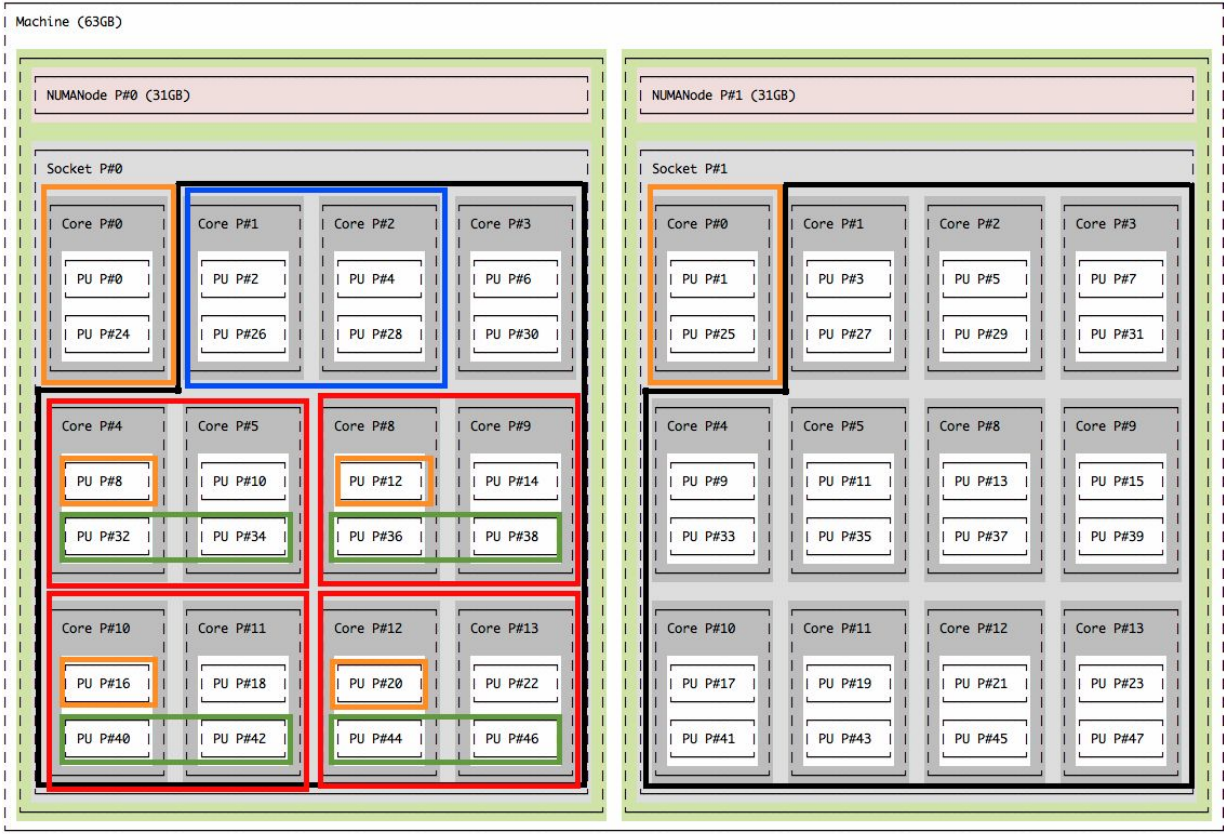
Agilio OVS - Example tuning setup (3/5)

CPU allocation design

- Host OS: (0,24), (1,25)
- Isolate all except Host OS
- VirtIO relay: (2,26), (4, 28)
- VM00: (8, 32), (10, 34)
- VM01: (16, 40), (18, 42)
- VM02: (12, 36), (14, 38)
- VM03: (20, 44), (22, 46)

VM specifications

- 4x CPUs
- 4GB RAM
- 2x VirtIO interfaces
- DPDK l2fwd app

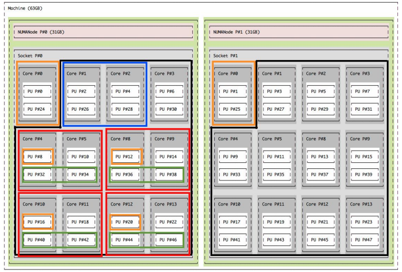


Agilio OVS - Example tuning setup (4/5)

Host configuration:

```
SDN_VIRTIORELAY_PARAM="--cpus=2,26,4,28 --virtio-cpu=0:2 --virtio-cpu=1:26
--virtio-cpu=2:2 --virtio-cpu=3:26 --virtio-cpu=4:4 --virtio-cpu=5:28
--virtio-cpu=6:4 --virtio-cpu=7:28 <<snippet>>"
```

```
# cat /proc/cmdline
intel_iommu=on iommu=pt hugepagesz=2M hugepages=16384 default_hugepagesz=2M
hugepagesz=1G hugepages=8 isolcpus=2-23,26-47 intel_idle.max_cstate=0
processor.max_cstate=0 idle=mwait intel_pstate=disable
```



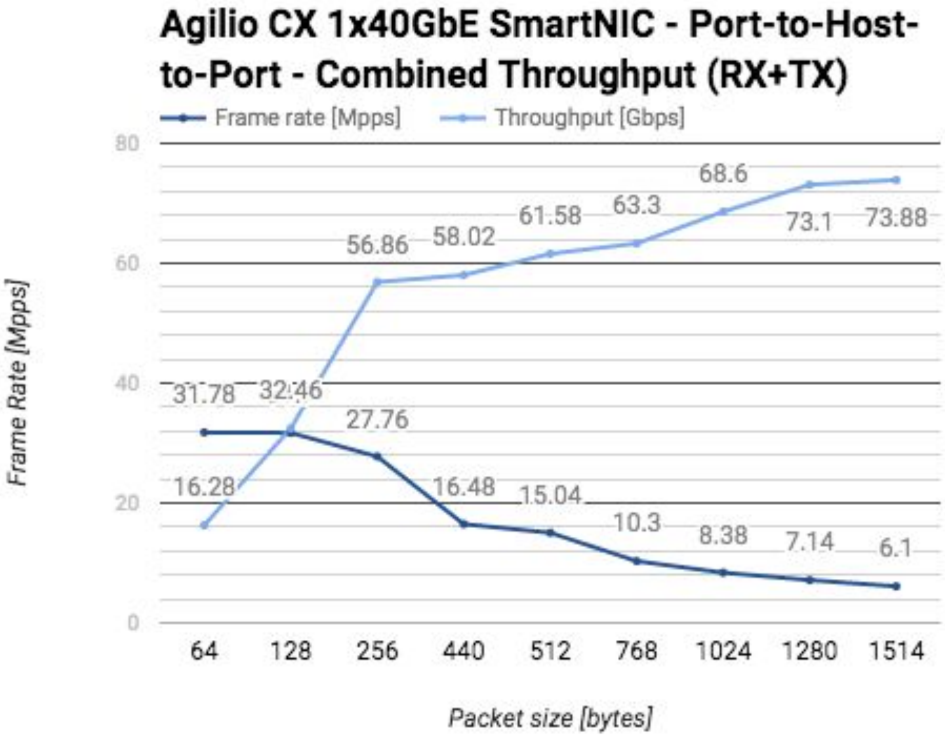
VM configuration:

```
service irqbalance off
DPDK applications inside VM:
./l2fwd -l 2-3 -n 4 — -p 3 -T 0
```

Agilio OVS - Example tuning setup (5/5)

Use Case

- Agilio OVS
- Agilio CX 1x40GbE SmartNIC
- SR-IOV
- External Traffic Generator: Ixia
- VM application: l2fwd



Contents

- Preliminaries
  - Equipment
  - Traffic generators
- Test topologies
- Host and VM configuration
  - NUMA Architecture
  - CPU allocation
  - BIOS
  - Host boot settings
  - Interrupt management
- Examples
- **Conclusion**

Conclusion

- What component are you trying to measure? What measurements needed?
- Data rate may dictate equipment and test bed configuration
  - 1G easy
  - ...
  - 100G hard
- Plan and provision test bed
- Isolate components from each other, and from external interference
  - Very important!
  - Partition and isolate CPUs
  - Turn off power management
  - Allocate memory
  - Eliminate extraneous interrupts, or steer to non data path CPUs

See also

1. Configuring HP ProLiant Servers for low-latency applications
   a. https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/c01804533-2014-may.pdf
2. Red Hat Performance Tuning Guide
   a. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html-single/Performance_Tuning_Guide/
3. Performance tuning from FD.io project
   a. https://wiki.fd.io/view/VPP/How_To_Optimize_Performance_(System_Tuning)
4. NUMA Best Practices for Dell PowerEdge 12th Generation Servers
   a. http://en.community.dell.com/techcenter/extras/m/white_papers/20266946

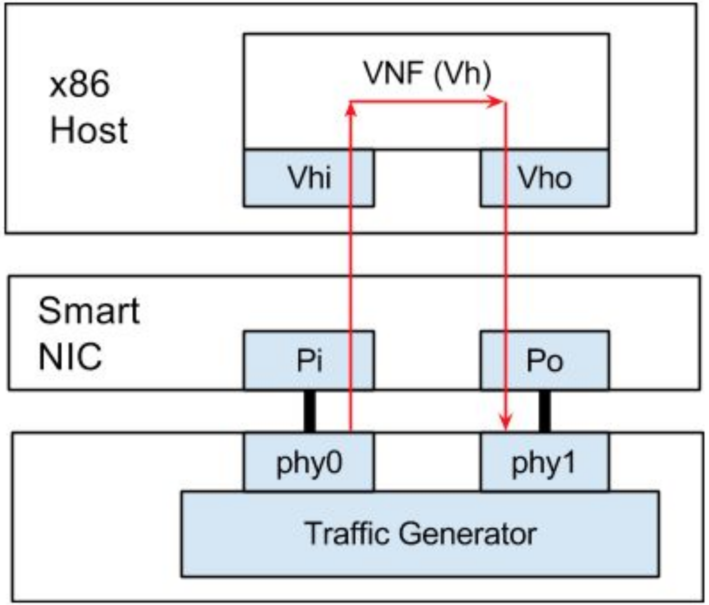
Reference and Backup slides

- More detailed slides, for reference

Test topologies - Port-to-Host-to-Port

Single DUT

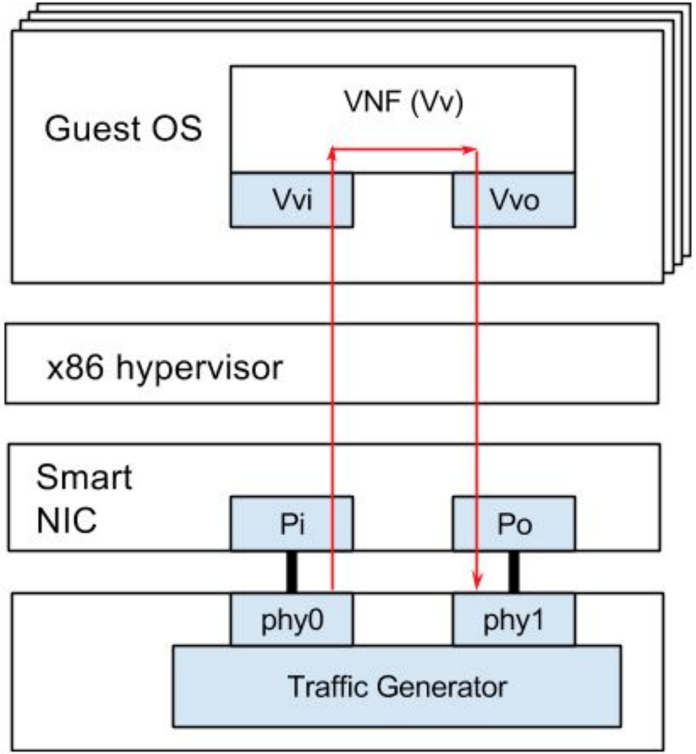
- L2 or L3 forwarding of packets between ports
- Ixia or other hardware traffic generator
- Host or VM based VNF
  - **Less complex and less overheads with host VNF**
  - More faithful to real use cases with VM
- Choice of VNF. Examples:
  - Simple: l2fwd, linux bridge
  - Real application: Snort, DPI, Caching
- What do you want to measure?
  - SmartNIC dataplane
    - Will need to ensure host OS, VM and VNF are not the bottlenecks
  - VNF
    - Usually easier to hit VNF performance limits.



Test topologies - Port-to-VM-to-Port

Single DUT

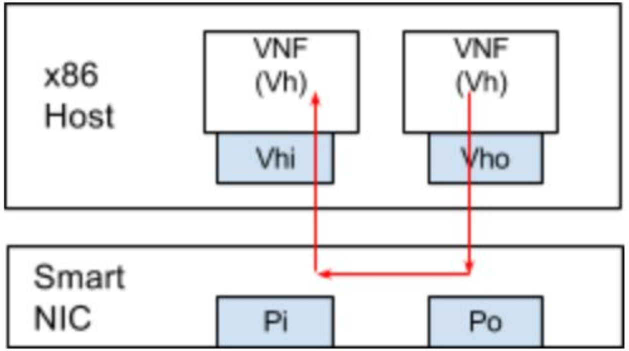
- L2 or L3 forwarding of packets between ports
- Ixia or other hardware traffic generator
- Host or VM based VNF
  - Less complex and less overheads with host VNF
  - **More faithful to real use cases with VM**
- Choice of VNF. Examples:
  - Simple: l2fwd, linux bridge
  - Real application: Snort, DPI, Caching
- What do you want to measure?
  - SmartNIC dataplane
    - Will need to ensure host OS, VM and VNF are not the bottlenecks
  - VNF
    - Usually easier to hit VNF performance limits.



Test topologies - Host-to-Host

Single DUT

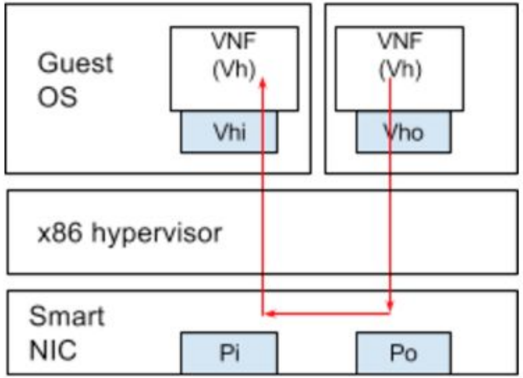
- No hardware traffic generator required.
- Single host with SmartNIC
- Use your choice of software traffic generator
  - Moongen (DPDK based)
  - DPDK-pktgen
  - Netperf
  - iperf
- Traffic generator will greatly affect configuration
  - DPDK based generators offer high performance with sufficient vCPUs and memory
  - Netperf/iperf offers simplicity but low performance. Must run multiple instances for useful results.
- Direct connect or switch between SmartNICs
- Less desirable than using a hardware based traffic generator configuration, but much more available



Test topologies - VM-to-VM

Single DUT

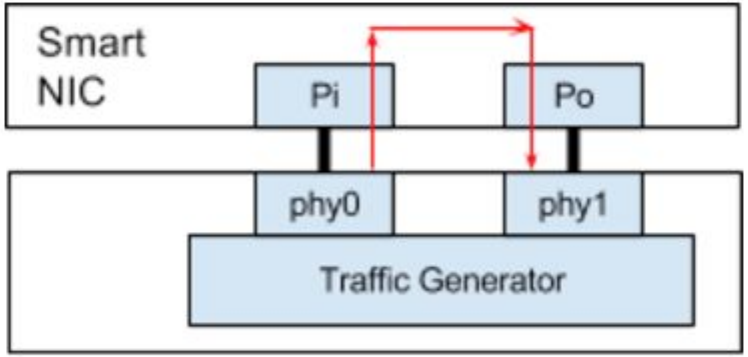
- No hardware traffic generator required.
- Single host with SmartNIC, 2 or more VMs
- Less desirable than using a hardware based traffic generator configuration, but much more available
- Use your choice of software traffic generator
  - Moongen (DPDK based)
  - DPDK-pktgen
  - Netperf
  - iperf
- Traffic generator will greatly affect configuration
  - DPDK based generators offer high performance with sufficient vCPUs and memory
  - Netperf offers simplicity but low performance. Must run multiple instances for useful results.
- Resource allocation and isolation essential (host OS, VMs, traffic generators) for valid results



Test topologies - Port-to-Port

Single DUT

- Useful for *direct* SmartNIC benchmarking purposes
    - Traffic does not pass PCIe bus
    - No host optimisation necessary
    - Host resources does not play a factor
    - Results without VNFs
- External traffic generator required (hardware or software)
- Can be achieved in a number of ways
    - Directly programming rules (wire application)
    - Priming flows (offload functionality)

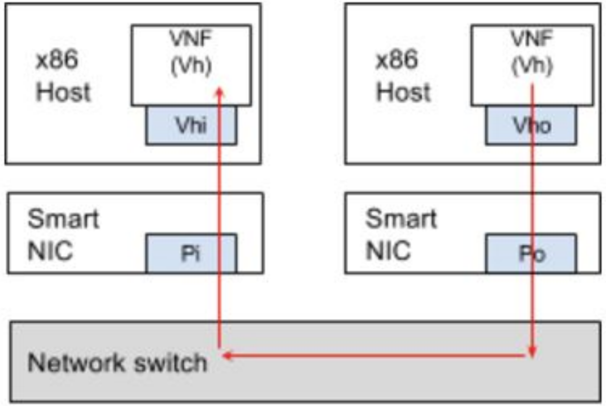


```
/opt/netronome/bin/ovs-ctl configure wire from sdn_p0 to sdn_p1
/opt/netronome/bin/ovs-ctl configure wire from sdn_p1 to sdn_p0
```

Test topologies - Host-to-Host

Dual DUT

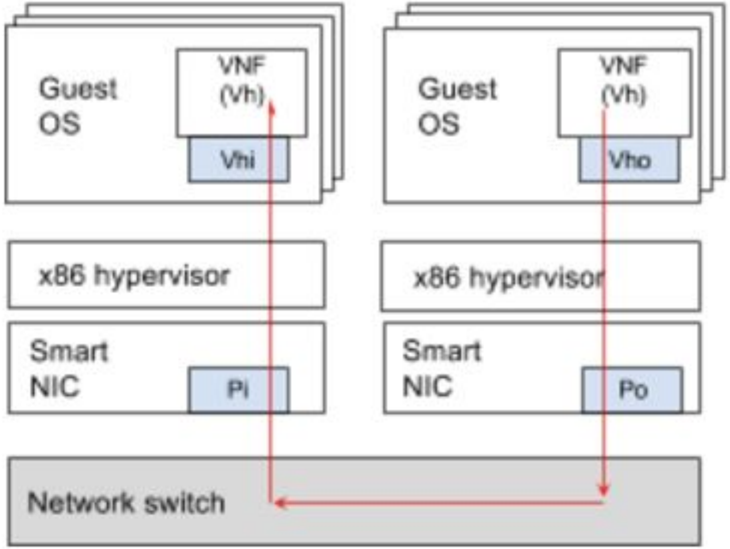
- 2 hosts connected via a switch.
- No VMs
- Traffic generators
  - Moongen
  - DPDK-pktgen
  - Netperf / iperf
- Physical port speed will be upper throughput limit for large packets.
- Less resource constrained configuration, easier to characterize SmartNIC performance
- Exercises full data path through the SmartNIC



Test Topologies - VM-to-VM

Dual DUT

- Traffic generators in VMs
  - one or more VMs on each host
- Resource allocation and isolation essential for valid results
- Larger number of VMs requires attention to vCPU allocation and NUMA


Guest OS
VNF (Vh)
Vhi
Guest OS
VNF (Vh)
Vho
x86 hypervisor
x86 hypervisor
Smart NIC
Pi
Smart NIC
Po
Network switch