Benchmarking and tuning NFV, using Yardstick/NSB, OVS, prox, perf, Intel PCM

Alexander Komarov

Application Engineer, Intel Software and Services Group

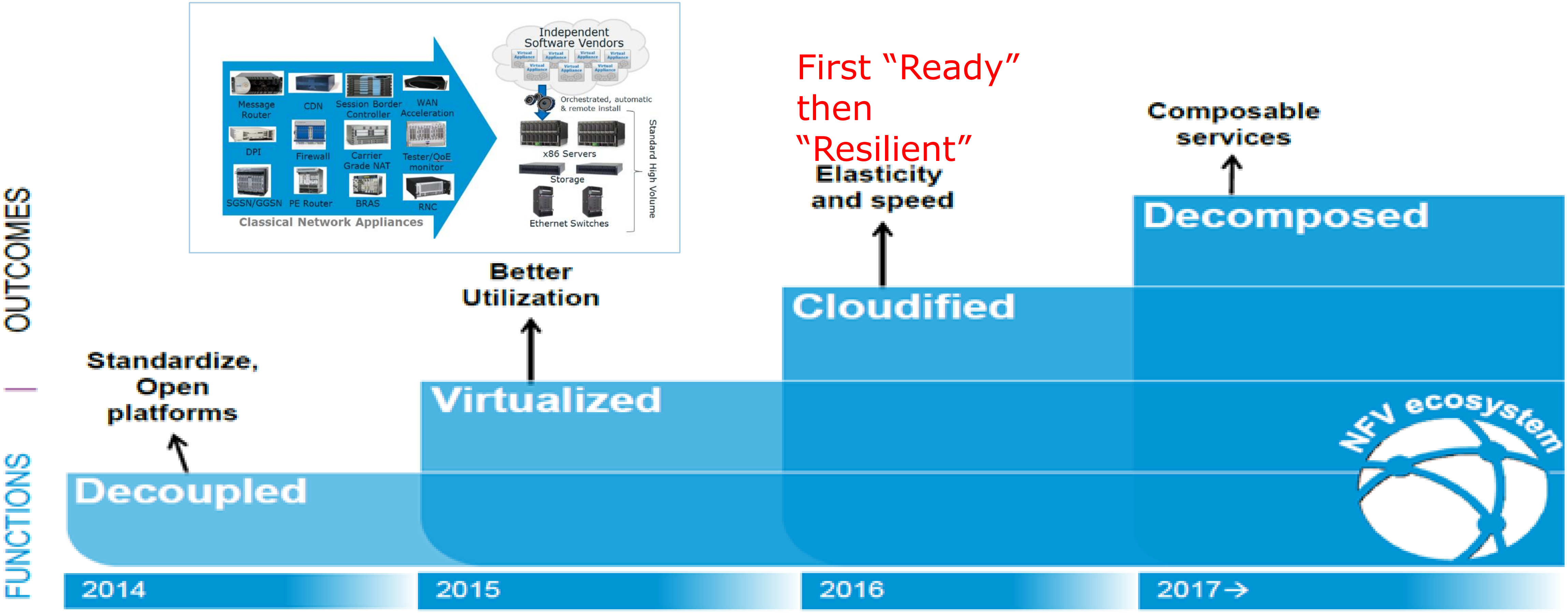
3.11.2017

Legal Disclaimer & Optimization Notice

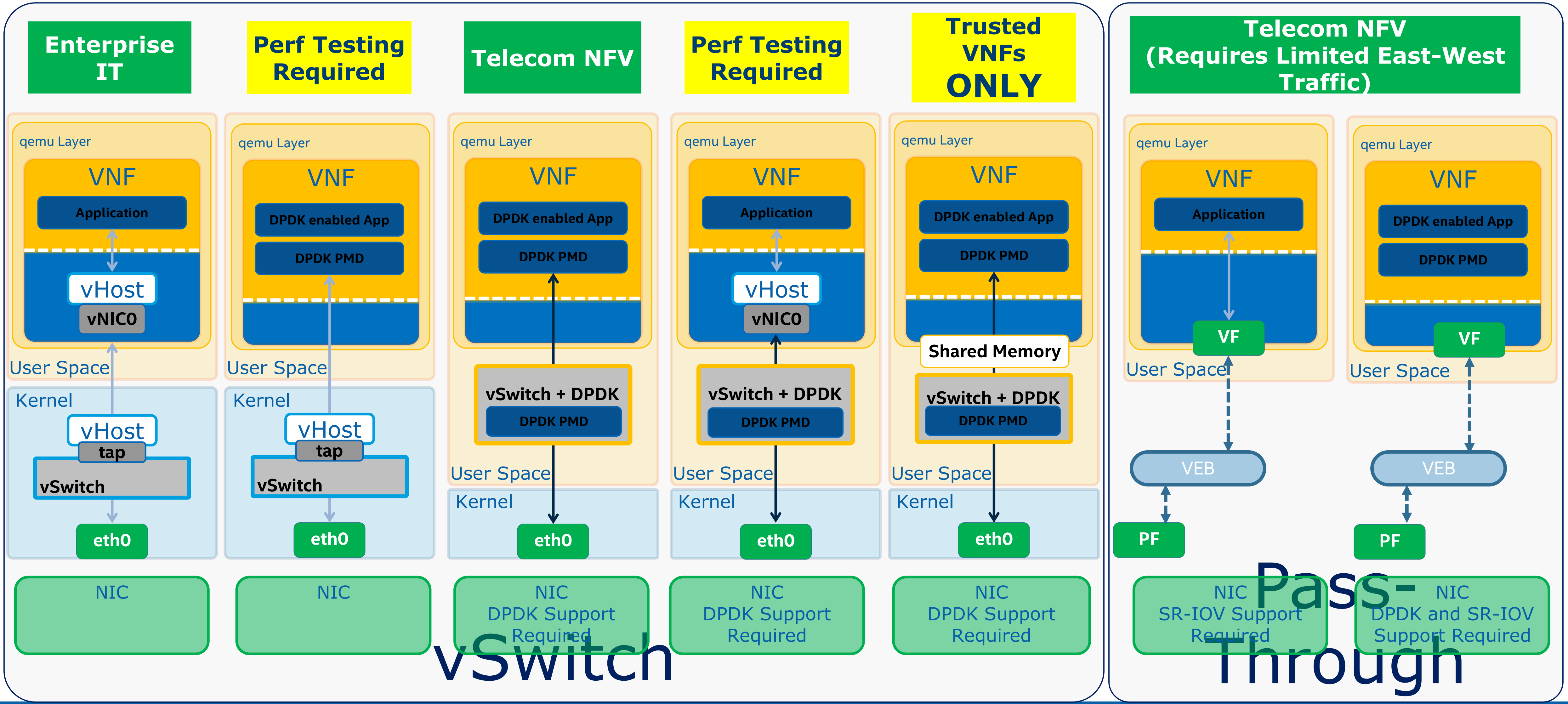
Agenda

- NFV intro
- NSB (Prox, Yardstick)
  - Benchmarking
  - Examples (mini case studies)
- Virtualized performance profiling
  - Profiling guest
  - Profiling in host
- Utilizing best instruction sets in private clouds (EPA or bottom up approach
- VNFs in containers vs VMs.

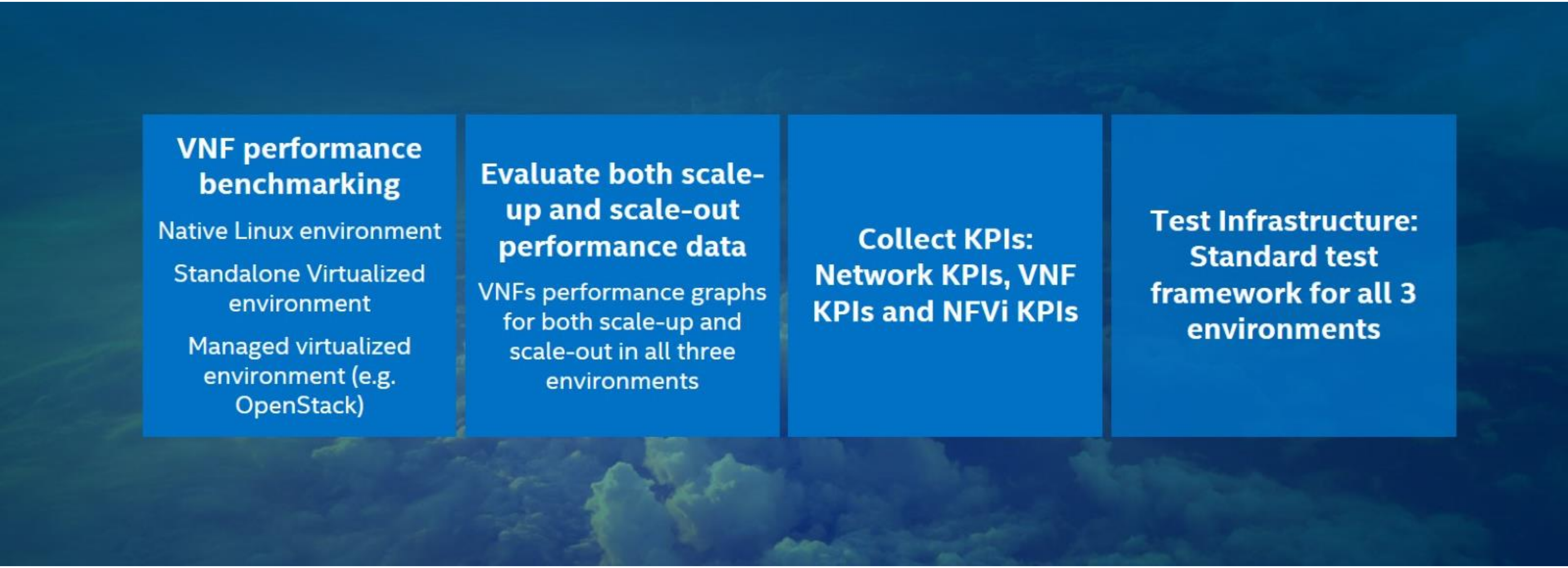
NFV software transformation



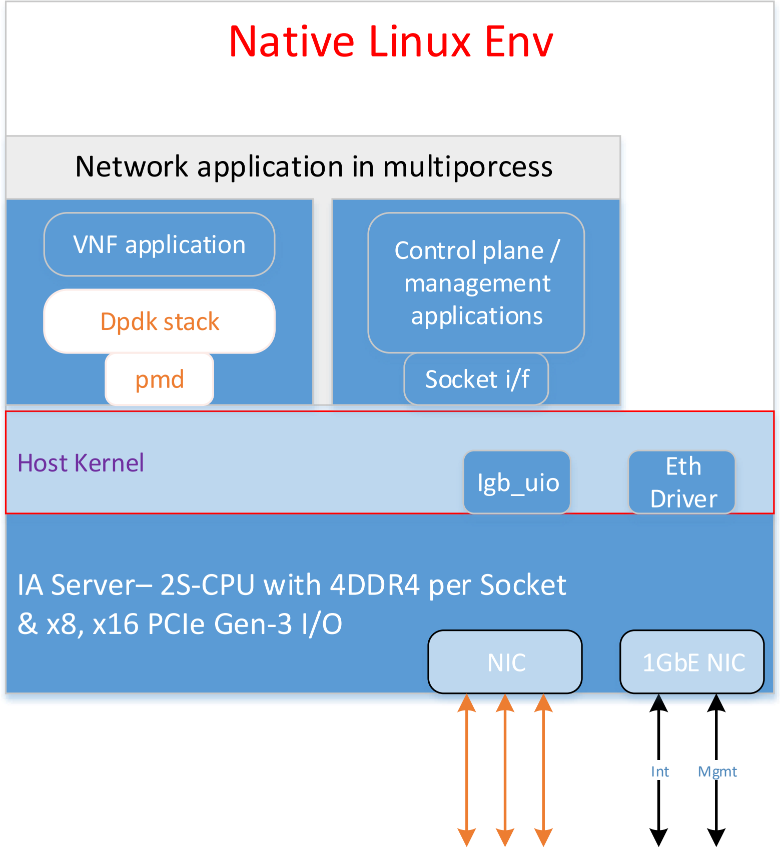
NFV config options for Networking Data Path

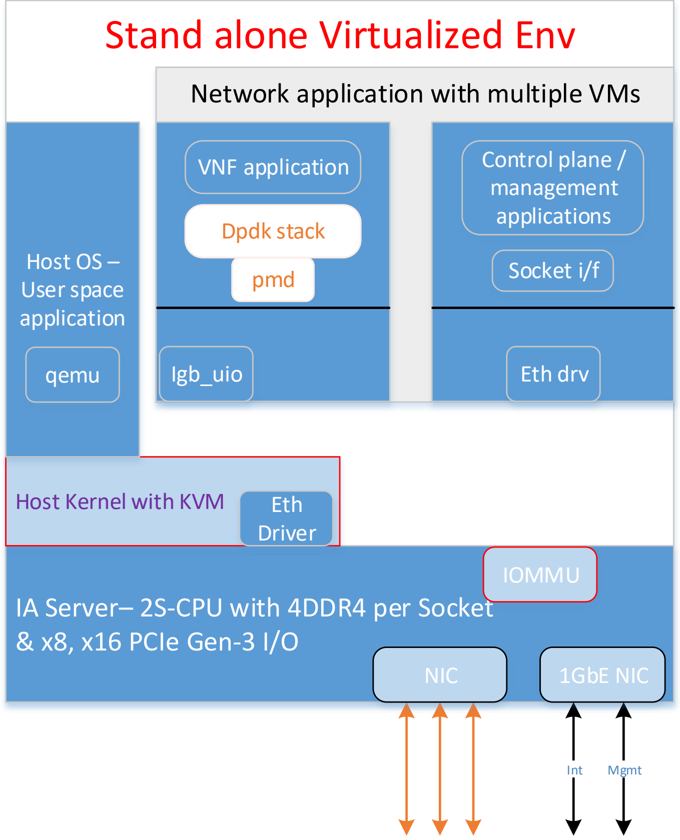


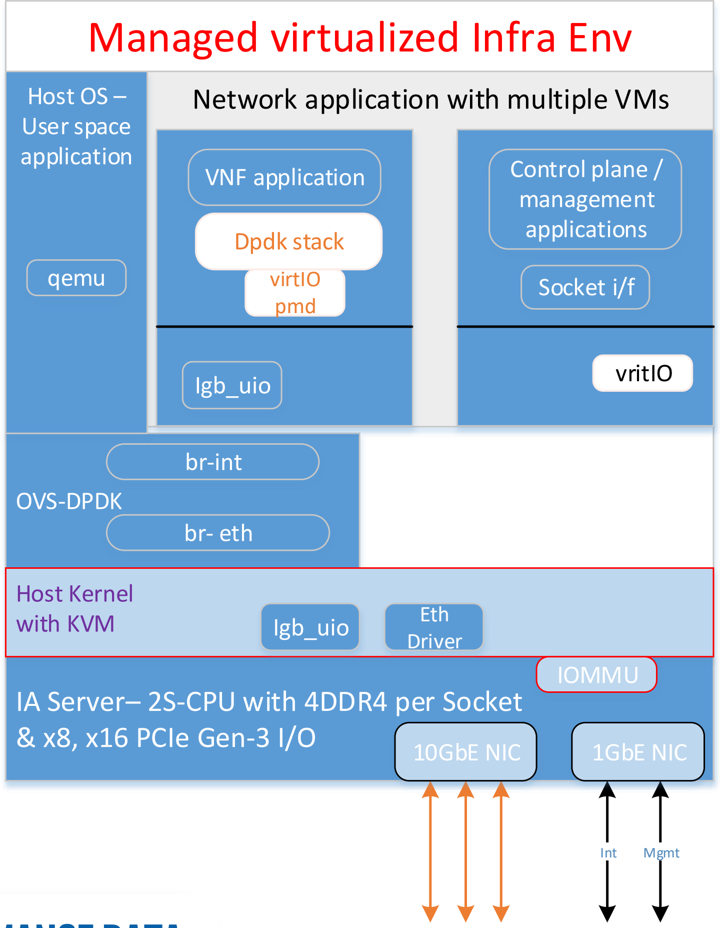
NSB Methodology – vnf performance benchmarking



Methodology: Vnf comparison through performance benchmarking







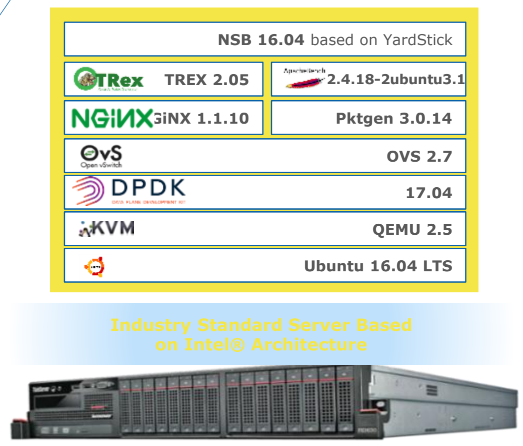
EVALUATE BOTH SCALE-UP AND SCALE-OUT PERFORMANCE DATA

Physical Network Topology and SW BOM

FOR SW BASED TRAFFIC GENERATORS

NETWORK TOPOLOGY

SW BOM



Supported traffic generators

- TREX
- Pktgen
- Ping
- Prox gen
- Benchmark/NGiNX

NFVI Characterization Tools - PROX

Workloads: developed in collaboration with SP partners

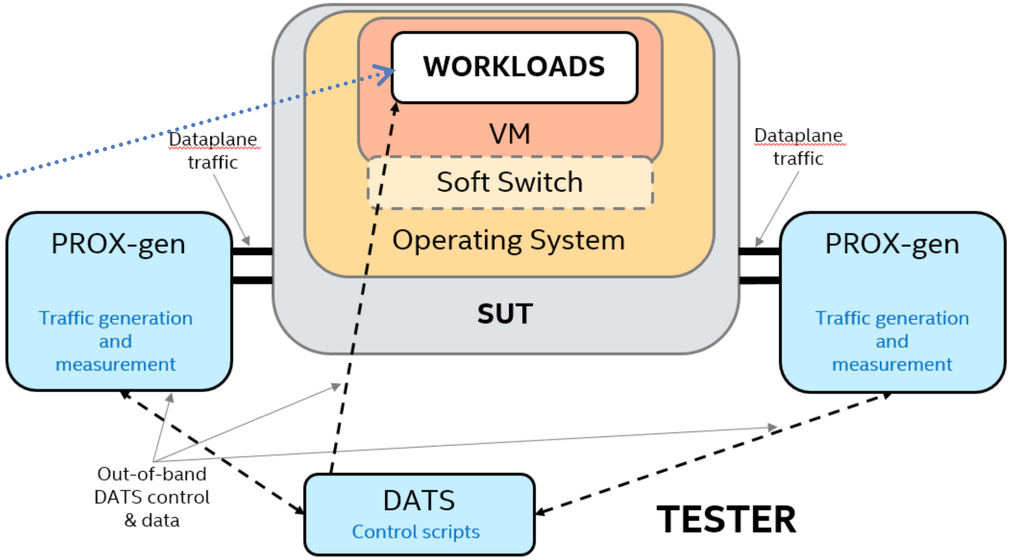
- Workloads exercise NFVI HW and SW features identified as performance-critical for VNFs and SFCs
- Workloads implement "kernels" of VNF/SFC functionality
- Run in PROX using different configuration files

Traffic generation and measurement (PROX-gen)

- Generates traffic that is specific to each workload
- Packet size and arrival distribution is configurable
- Automation interface provided for DATS

DATS: Dataplane Automated Testing System

- Scripts to automatically drive, measure, report multiple dataplane test cases.
- Measurements with 0 packet loss (according to RFC2544)
- Workload-specific Key Performance Indicator (KPI) used as summary statistic
- Output is an automatically generated report
- Enables faster, more reliable test case execution



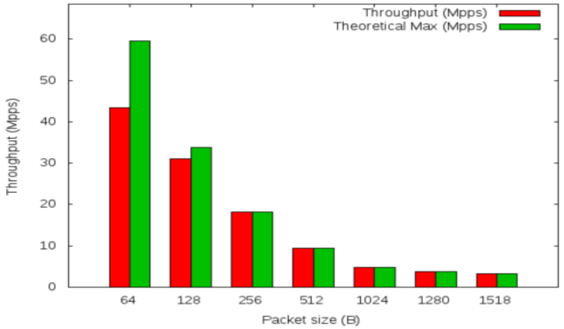
Port forwarding without touching packets

Description

The application will take packets in from one port and forward them unmodified to another port. This use case is not representing any *real* use case but it is a good start to do a sanity check of the environment.

The KPI is the number of packets per second for 64 byte packets with an accepted minimal packet loss (zero, no packet loss).

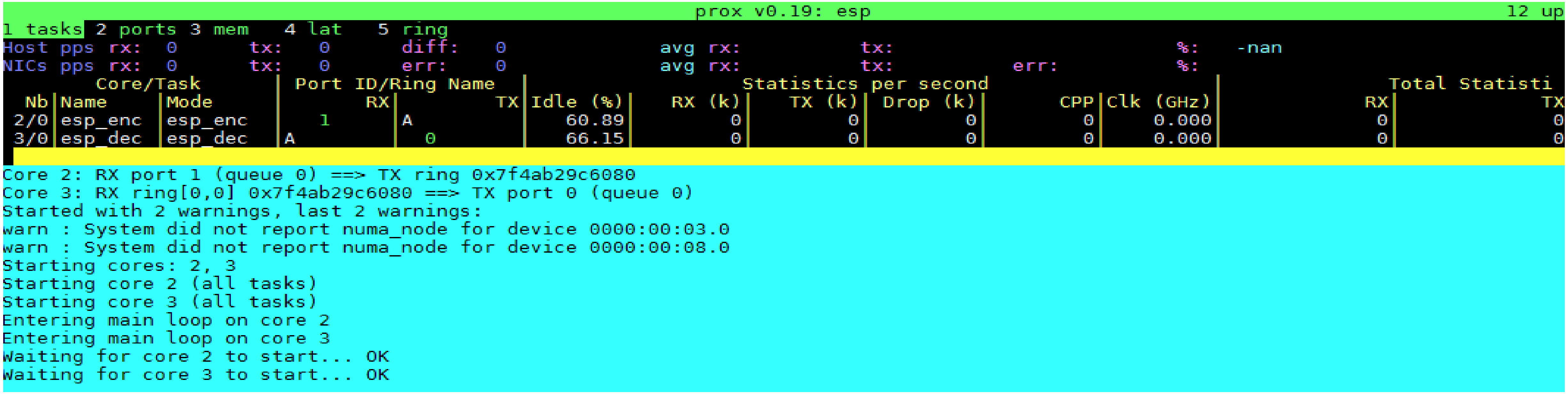
Result



Packet size (B)	Throughput (Mpps)	Theoretical Max (Mpps)	Duration (s)
64	43.3945123314	59.5238095238	120.278173923
128	30.9998391896	33.7837837838	120.292181015
256	18.1156148588	18.115942029	15.035173893
512	9.39842704915	9.3984962406	15.0372900963
1024	4.78927200203	4.78927203065	15.0380678177
1280	3.84615146073	3.84615384615	15.0364170074
1518	3.25097532563	3.25097529259	15.0354731083

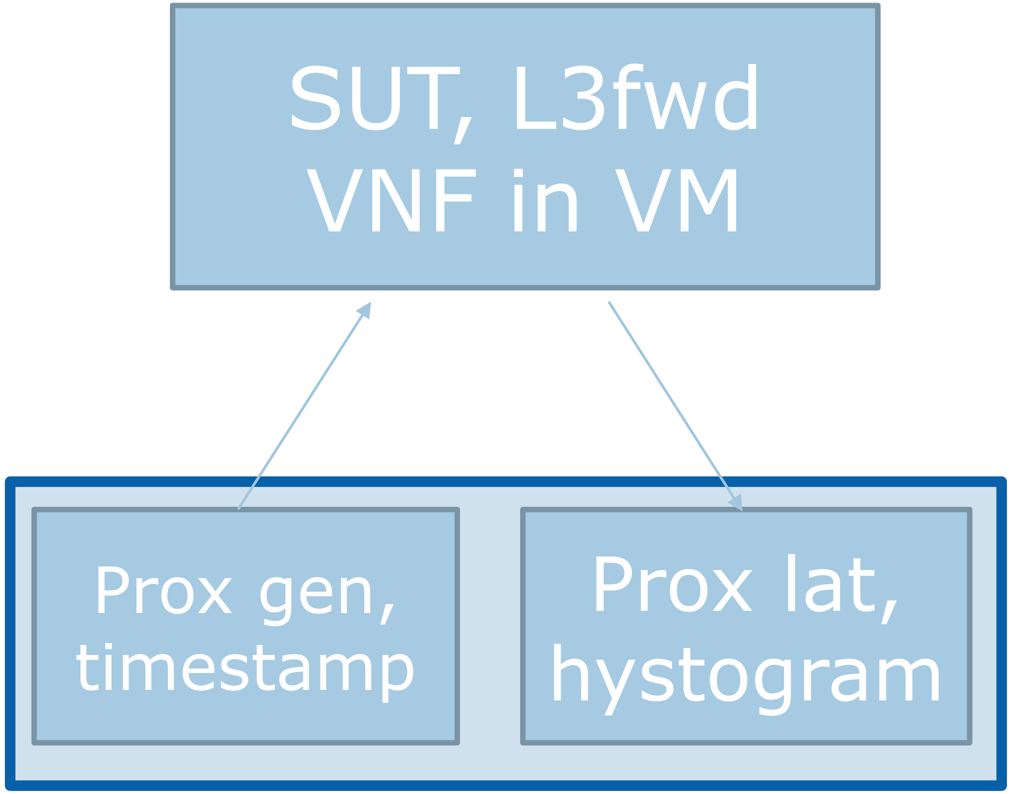
Prox screenshot

- Open source project at 01.org, github (under OPNFV)
- On top of DPDK and prox, developer can combine building blocks in a text config file to create DPDK performance demonstrators/custom traffic gen/test tools
- Developer can implement additional building blocks (complimenting available blocks: Gen,lat,nop,acl,ipsec,qos,qinq,classify,cgnat,gre,route,police,lb*,etc)
- Convenient NCURSES GUI for live stats monitoring / configuration



Prox case study 1

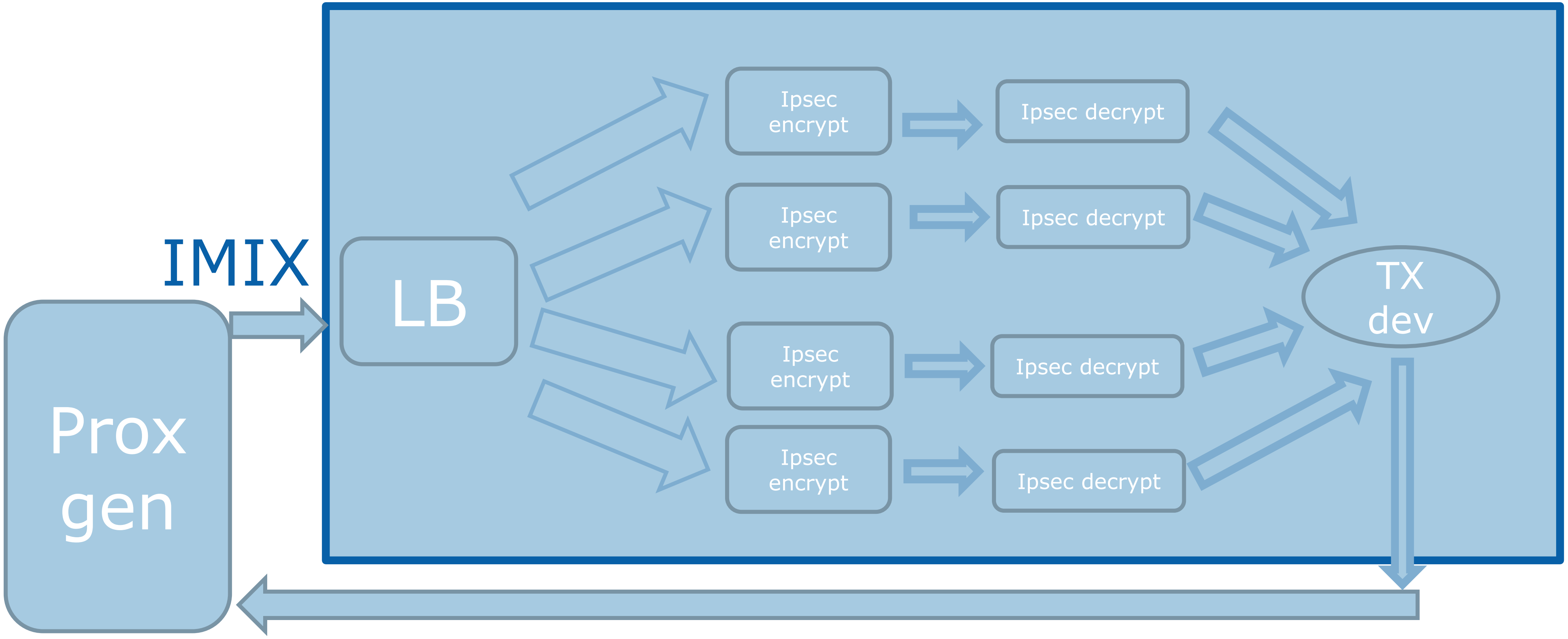
- VNF Latency PCI-Passthrough
- VNF Jitter PCI-Passthrough
- VNF Latency SR-IOV
- VNF Jitter SR-IOV



Results, µs:

	10M small packets (64 bytes) l3 fwd VNF			
	simple fwd, native	l3fwd, native	l3fwd, PCI passthrough	l3fwd, SR-IOV
min	14	30	45	40
avg	30	64	74	69
max	90	145	160	160

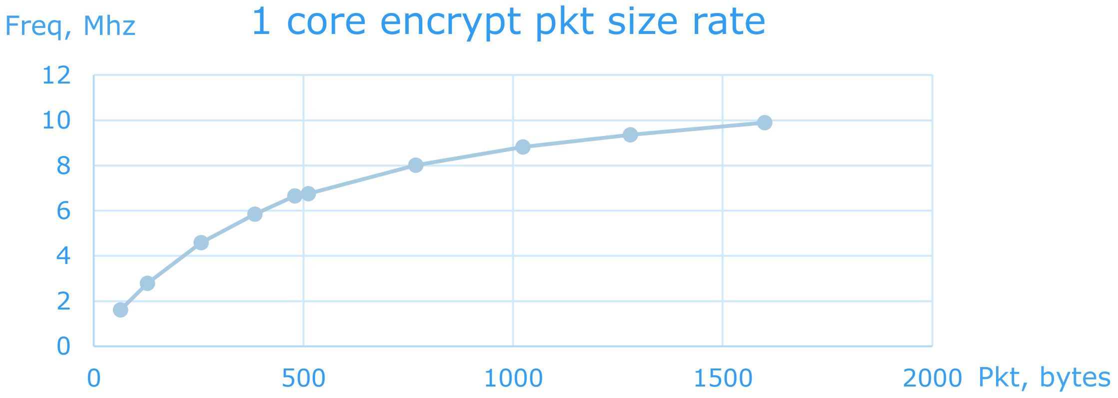
Case study 2 – Prox setup



Case study 2 – SKL-SP performance

- Xeon Platinum 8168 @2.7Ghz, RHEL 7.2, Niantic NICs.
- ~2X performance improvement over BDW-EP on IPSec workload: ~20% from IPC increase, ~80% from SIMD optimizations in DPDK-17.05 and Intel multibuffer crypto library.
- Mhz/Kpkt, @IMIX pkt rate:

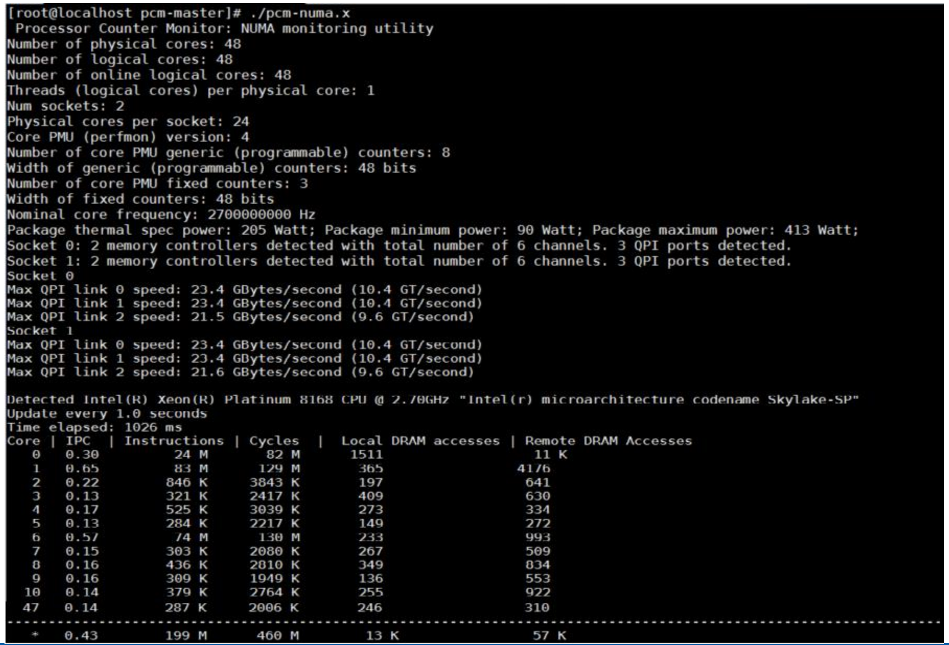
BDW-EP	SKL-SP
3.4	1.5

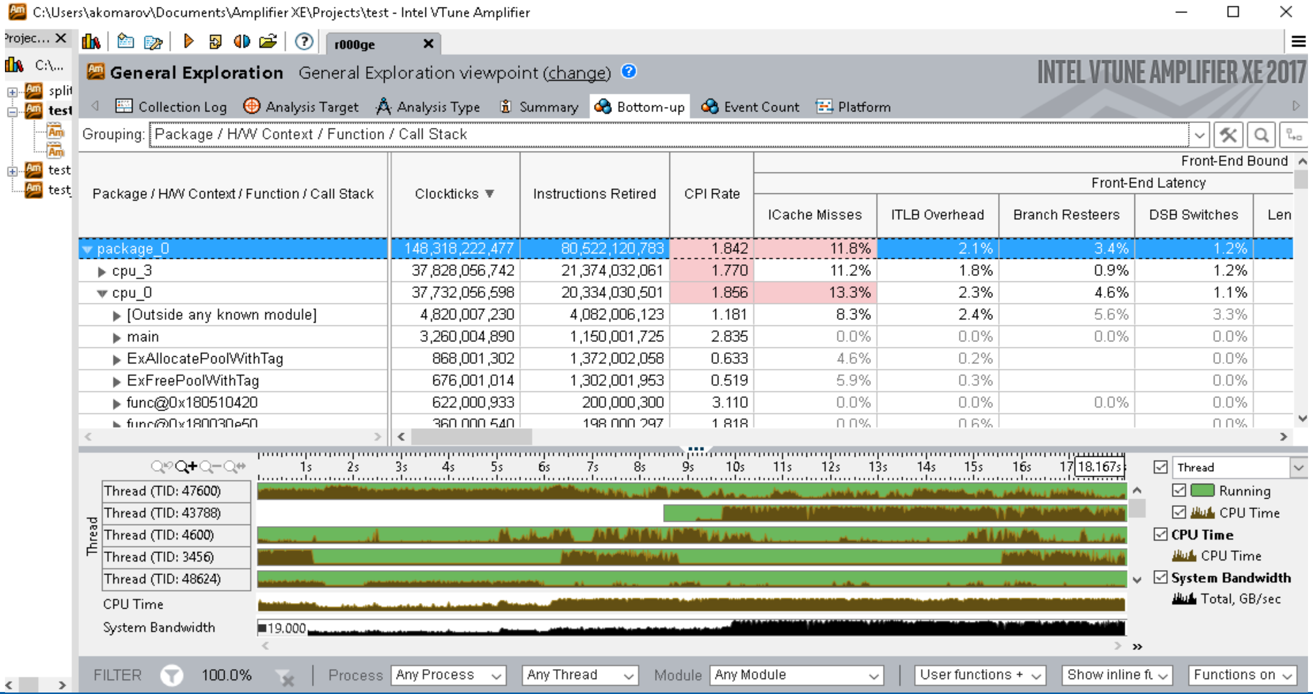


- At IMIX full line rate (3940Kpkt), need 2 encrypter cores (+ LB core), 2 decrypter cores. Scales very well with MHZ and cores.
- When using integrated QAT, need 1 core for encrypt and 1 core for decrypt

CPU/system profiling tools

- Linux perf – de facto standard now. PMU sampling in NMIs
- Intel PCM – PMU counting mode, uncore PMU
- Intel Vtune – standalone or as perf results viewer. Best viewer for x86 PMU sampling. Custom collectors with IPT support, stack sampling support; or frontend for perf





Profiling in host

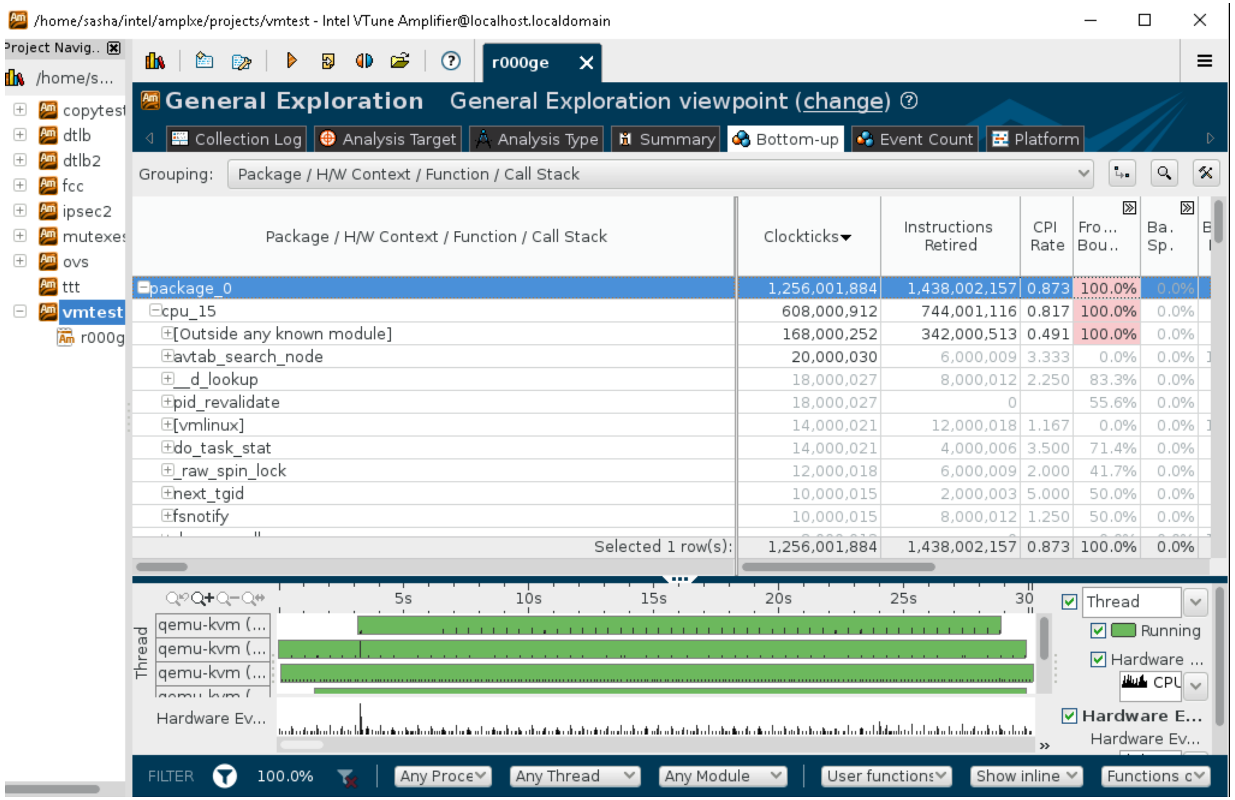
- # perf kvm --host –guest … -a record –o perf.data
  - Get /proc/kallsyms, /proc/modules from guest first.
  - Read results with perf or Vtune, or collect with Vtune (will invoke perf kvm collector).
- # Intel pcm-core, pcm-pcie, pcm-memory, pcm-numa, pcm-tsx, pcm-power, pcm-sensors.
  - Watch for excessive NUMA, memory, PCI traffic
- Any way, you only see detailed host data,

kernel guest data [single guest!],

and aggregated user mode guest data.

- Why bother - ?

  Because we can read and sample all cores and uncore PMU counters!



Profiling in guest

#perf list // Run in guest

- L1-dcache-loads [Hardware cache event] GOOD
- Only cpu-clock,etc [Software event] BAD. Make sure qemu –cpu host, OpenStack config capabilities:vcpu_model.features=arch_perfmon

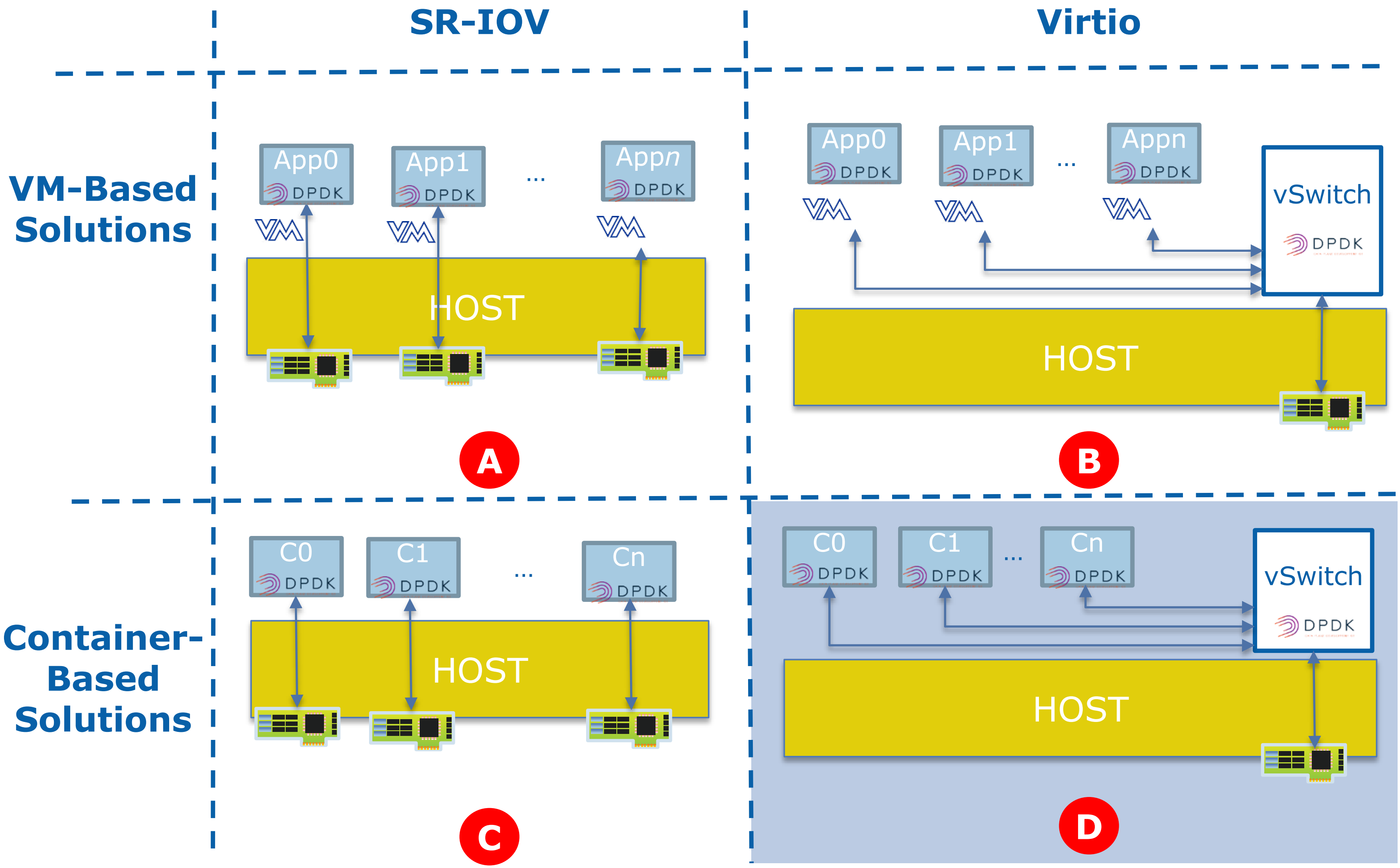
perf record –a –e instructions,ref-cycles,LLC-loads,LLC-load-misses,LLC-stores,LLC-store-misses,… sleep 60

- Open results (perf.data) in perf viewer or import to Vtune (rename to *perf).
- You don't see all PMU events – KVM only passes a dozen most important ones to guests
- All time sampling profiling methods would also work..

Utilizing best ISA in guests

- VNFs (Virtual Network Function) vendors have to deploy to private clouds, often configured to spoof CPUids (Sandy Bridge as default is common). So they have to run with “least common denominator” architecture settings, that is not very efficient.
- Enhanced Platform Awareness approach – top down, configure guests with real CPUIDs, use orchestration to start binaries compiled for the best supported instruction set.
- Bottom up approach – use a small (in LOC) tool that reveals a physical CPU model when running under a hypervisor that hides/spoofs real CPUID to guests.
- Run a tool, get a real physical CPU model, select the right binary (with AVX, AVX2, AES-NI, AVX-512 support)

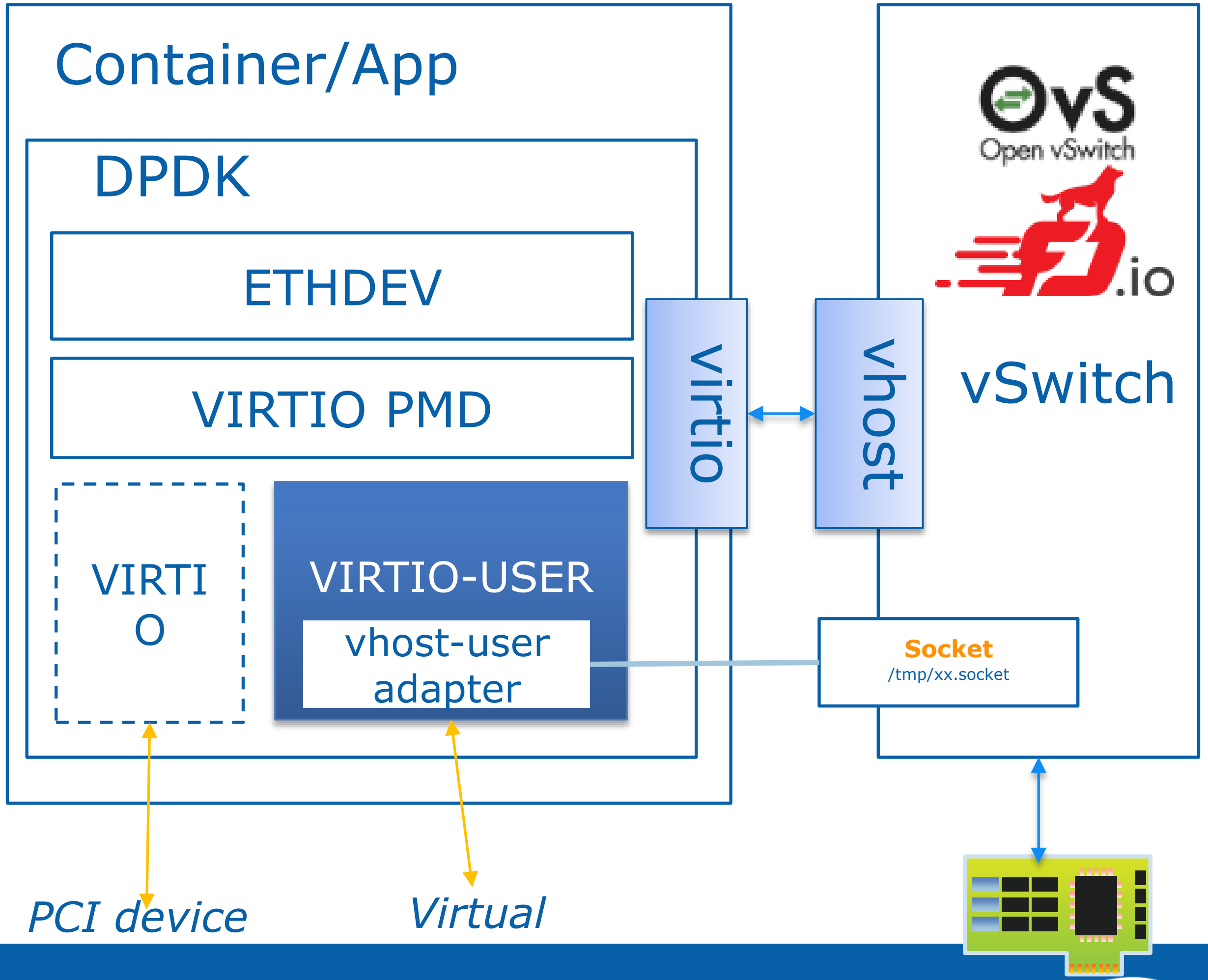
VM & Container Usage Models in NFV



Virtio in Containers

Description

- Virtio in Containers is a new approach to high-speed networking for containers.
- In a VM, QEMU helps with device emulations and interaction with the backend.
- In containers, we don't have QEMU:
  - We could introduce a kernel module to fulfil the same function as QEMU, but we're already trying to remove the existing out-of-tree kernel modules from DPDK.
  - Instead, all of the work is done in the DPDK PMD driver. We present virtio as a virtual device, just like the way that Ring, PCAP, or other virtual devices are used in DPDK. The control message are also handled through the DPDK driver.



Download links, docs

- Intel PCM - http://www.intel.com/software/pcm
- NSB/Yardstick - https://wiki.opnfv.org/display/yardstick/Yardstick
- Prox - https://01.org/intel-data-plane-performance-demonstrators
  - Source, as a part of OPNFV: https://github.com/opnfv/samplevnf/tree/master/VNFs/DPPD-PROX
- Cisco Trex - https://trex-tgn.cisco.com/
- Linux perf - https://perf.wiki.kernel.org/index.php/Main_Page
- Intel Vtune - https://software.intel.com/en-us/intel-vtune-amplifier-xe/
- CPUvirt2phys - https://software.intel.com/en-us/articles/how-to-tell-cpu-model-when-running-under-hypervisor-that-spoofs-cpuid

intel®