WHITE PAPER

Communications Service Providers
Virtual Router

High Performance Packet Processing on Intel® Architecture Platforms: Brocade* 5600 vRouter

This test report documents the performance obtained by running the router functionality of the Brocade* 5600 vRouter in a virtual environment on a dual-socket Intel® Xeon® processor–based server.

Executive Summary

Today, most communications service providers (CommSP) use in their networks many proprietary, fixed-function hardware platforms to deliver services. Adding new services/functions to a network usually requires new equipment dedicated to the new functions. CommSPs find such an approach of deploying new services slow, costly, and difficult to scale and manage.

In an effort to make their networks and services more agile to deploy and manage, many CommSPs are embracing network functions virtualization (NFV), a new service deployment architecture that enables network functions to run on virtual machines on Intel® architecture–based servers. Such servers are capable of hosting multiple applications, which reduces cost and power consumption. Using a virtualized environment, new software instances can be deployed or turned off very rapidly. Introducing NFV is expected to create much more flexible network services, to permanently lower network costs, and to improve time to market for a significant return on investment.

A big question in this transition is about whether the NFV system can perform at the same level as the dedicated appliance it replaces. Many hardware appliances have specialized packet processing ASICs that help to deliver wire speed performance for even the fastest network speeds. The challenge is to design an NFV system that can match the same or higher performance using Intel® Xeon® processors. In the NFV paradigm, a service comprises the software component called the virtual network function (VNF), the virtualization infrastructure (virtual machines, server), and the management and orchestration.

The Brocade* 5600 vRouter is a commercial network router VNF from Brocade Corp. The Brocade 5600 vRouter delivers advanced layer 3/4 routing, stateful firewall, NAT, and VPN capabilities in software that is designed for carrier-class reliability and performance. The routing functions built into a VNF make it a key piece of equipment in a CommSP network, both in the central office and as a component of a virtual enterprise customer premises equipment (vE-CPE) solution.

Given that the performance of the Brocade 5600 vRouter is crucial, Intel developed a testing methodology that can be used by Intel employees, partners, and customers to evaluate performance of the Brocade 5600 vRouter in their own networks.

Table of Contents

This document focuses on the performance (both throughput and latency) obtained by running the router functionality of the Brocade 5600 vRouter in a virtual environment on a dual-socket Intel Xeon processor–based server. The parameters used (number of routes, next hops) as well as the traffic pattern are fully described in this document. Other parameters and other traffic patterns might yield different results. In the rest of this paper, vRouter and Brocade 5600 refer to Brocade 5600 vRouter.

Figure 1 shows that the vRouter is capable of maintaining line rate using 4x 10 Gbps interfaces, when using packet sizes of 256 bytes or higher, and up to 32k next hops per interface.[1]

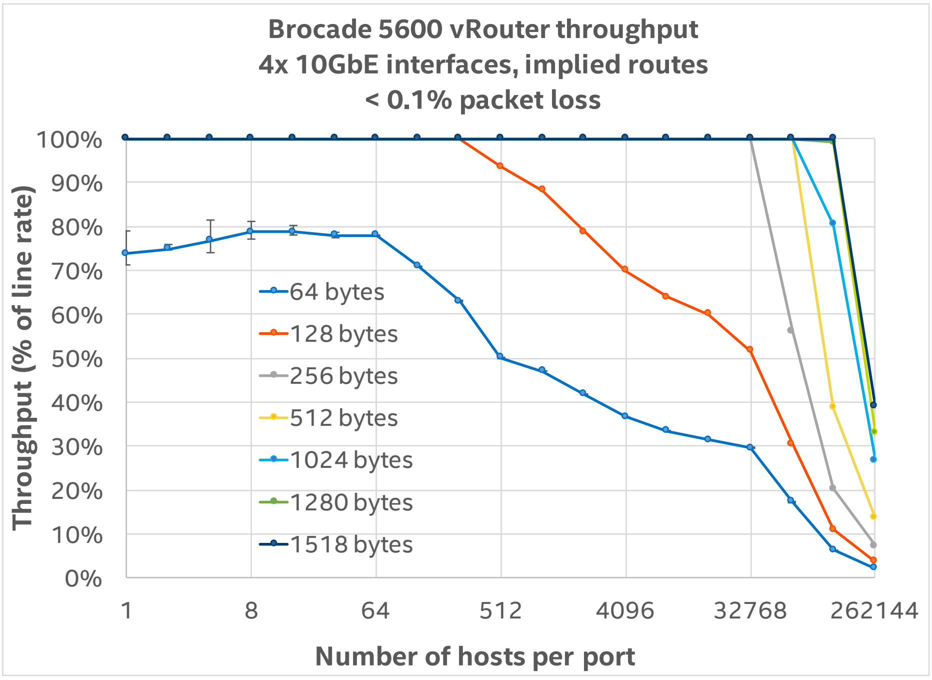


Figure 1. Infuence of number of next hops on performance[1]

As shown on Figure 2, the latency obtained for 64 bytes packets is between 20 and 30 microseconds when the load is between 1% and 50% of line rate.[1]

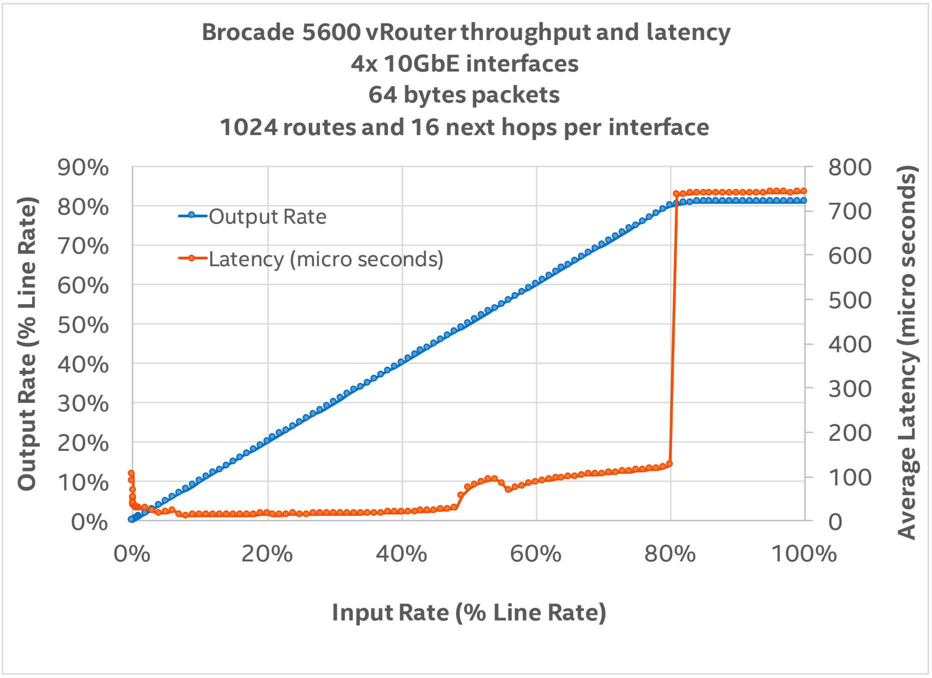


Figure 2. vRouter's latency and throughput in function of the input rate[1]

[1] Intel internal analysis. See Section 3 for the system under test's configuration details, and Section 4 for the test generators' configuraton details.

1 Test Purpose

We conducted the tests described in sections 1.2.1 to 1.2.5 to show the performance when one CPU socket of a dual-socket server is used (Figure 3). Section 1.2.6 gives some performance expectations for when the VNF is scaled to run one Brocade 5600 vRouter instance on each CPU socket (two instances in total).

The Brocade 5600 vRouter is running in a virtual machine, using QEMU[2] as the hypervisor and CentOS* Linux*[3] as the host operating system. PCI pass through is being used,[4] i.e., the control of the full physical device is given to the virtual machine; a virtual switch was not used in the fast path (see Figure 3).

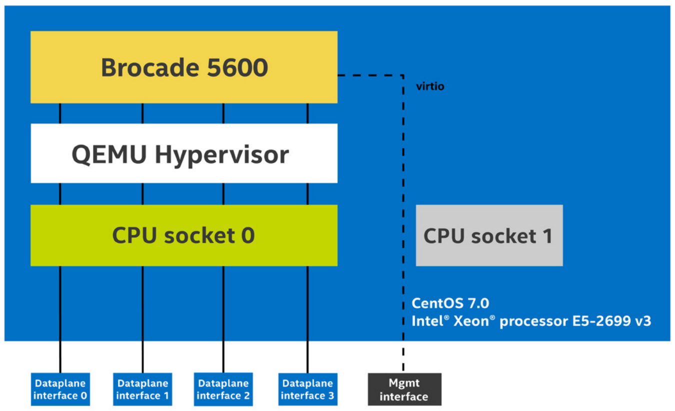


Figure 3. High level architecture

1.1 Test Setup Details

The Brocade 5600 vRouter is characterized under network load produced by test generators. The test generators generate IP traffic toward four 10 GbE interfaces (results in sections 1.2.1 to 1.2.5) or eight 10 GbE interfaces (sections 1.2.6), and they measure the traffic coming from those interfaces. Those test generators can be from commercial suppliers such as Ixia* or Spirent,* or Intel architecture–based servers running pktgen or prox, which are traffic-generation applications that are part of the Data Plane Development Kit (DPDK) library of applications.

For automation purposes, prox (https://01.org/intel-data-plane-performance-demonstrators/prox-overview) has been used to generate the traffic and to measure the throughput and latency from the vRouter.[5] An Ixia traffic generator was used as well to confirm some key data results.

One of the key parameters when characterizing the performance of a router is the number of active hosts/next-hops that can be directly seen by the router (in blue on Figure 4). The number of routes in the routing table is another key parameter: on Figure 4, the yellow and green networks can only be accessed by "Router 1" using the routes in the routing tables.

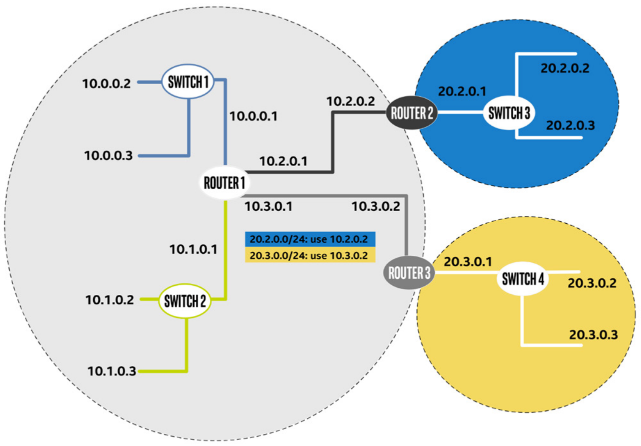


Figure 4. Network topology

Instead of the number of hosts, some reports sometimes use the number of flows. For an example, see "Brocade Vyatta 5600 vRouter: NFV Routing and Security Performance Benchmark on Mid-Range Cloud Servers."[6] While a flow often refers to 5-tuple (i.e., IP source and destination, UDP (or TCP) source and destination, and protocol), another definition of a flow is being used in that report (two tuples: IP source and IP destination), as it is expected that layer 4 fields (UDP source and destination ports, protocol) have no influence on a L3 router.

To avoid any confusion with the 5-tuple flows, we will use the term "host-pair" to refer to "two tuples."

Test generators (TG) simulate a certain number of hosts/next hops (one or two per interface shown on Figure 5) and also generate packet flows.

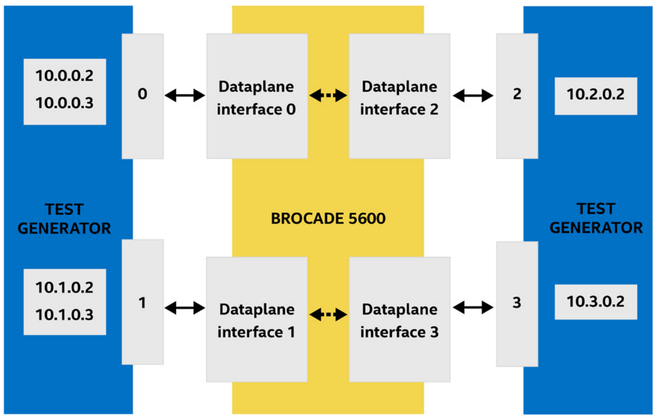


Figure 5. Test setup overview

Figure 5 shows the test setup used for this characterization, using two systems to generate traffic to four ports. In this setup, TG interface 0 (TG) only generates packets towards TG2; TG2 only generates traffic towards TG0, TG1 towards TG3, and TG3 towards TG1.

[2] http://wiki.qemu.org/Main_Page
[3] https://www.centos.org
[4] PCI-Pass-through was chosen to stay focused on CPU characteristics and not be distracted by vNIC/NIC capabilities and does not reflect on what the Brocade 5600 vRouter supports
[5] Choice of test generator is simply based on engineer's preference and has no known impact on the performance numbers.
[6] https://networkbuilders.intel.com/docs/Vyatta_5600_Performance_Test_Full_Report.pdf

In such a setup, if four hosts per interface are used, 32 bidirectional host-pairs are generated (4 x 4 x 2). If 1,000 hosts per interface are used, 2 million flows are generated (1000 x 1000 x 2). Different traffic profiles (e.g., TG0 generating traffic toward all TG ports) will deliver different performance.

1.2 Test Results

1.2.1 Throughput

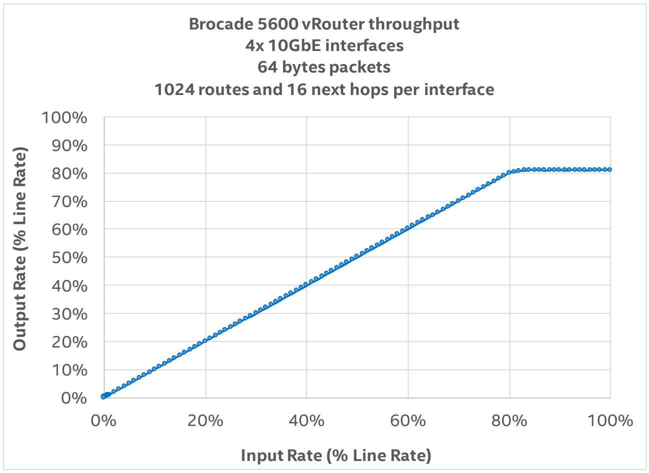


Figure 6. Brocade 5600 vRouter throughput[7]

The router is configured with 1024 routes and 16 next hops per interface.

This result (Figure 6) shows the throughput obtained while increasing the input rate, using 64 bytes packets.[7] We see that the output rate increases linearly with the input rate (as expected), up to around 80% of the line rate, where the output rates saturate. The output rate is limited to 80% of the line rate because of PCIe* Gen2 bandwidth limitation on Intel® 82599 10 Gigabit Ethernet Controller when using both interfaces.

The overload behavior of the vRouter is optimal because the output rate remains stable when system became overloaded (above 80% of input rate). When the router reaches this throughput level, it drops the extra packets, but it is still able to handle 80% of the line rate properly. A bad behavior would show an output rate decreasing when the input rate is higher than 80% of the line rate: the router would spend so much time dropping packets that it would not be able to route packets properly.

1.2.2 Packet Loss

The previous result demonstrated the outgoing throughput. In the ideal situation, a router should not drop packets until it becomes overloaded. Figure 7 shows the packet loss while increasing the input rate.

We see that at all input rates lower than 80% of the line rate, there are no dropped packets.[7]

A very small percentage of packet loss might sometimes happen at rates lower than 80%, and is usually due to a fastpath thread running the dataplane being interrupted.

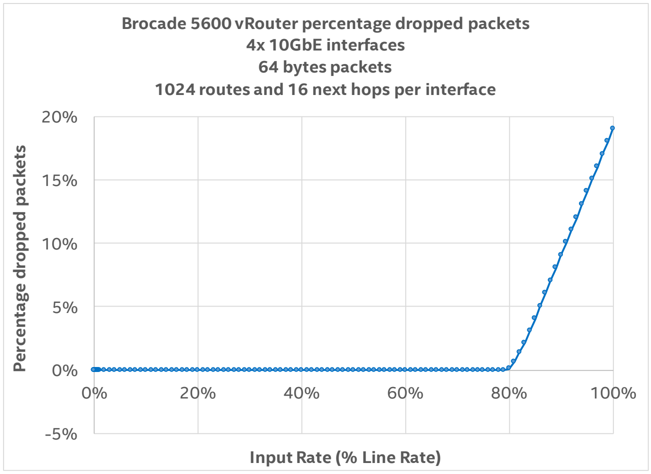


Figure 7. Brocade 5600 vRouter packet drop[7]

This behavior can usually be avoided by a better tuning of the host system (running QEMU), preventing all datapath-related cores from being interrupted.

At high input rates (> 80% of line rate), the dropped packet rate increases dramatically. This is expected as the system becomes overloaded. In previous tests (see Figure 6), we have seen that at those rates, the output rate saturates at 80% of the line rate, hence the extra packets (approximately the difference between the input rate and the output rate) are dropped.

1.2.3 Latency

Latency is a very important characteristic of a router VNF.

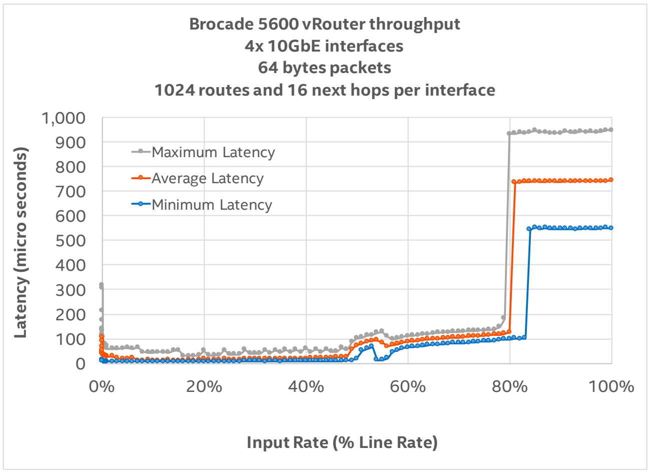


Figure 8. Brocade 5600 vRouter latency[7]

We see (Figure 8) that the average latency for most input rates is between 20 and 30 microseconds. Only for very low input rates (< 1%) does the latency increase to up to around 250 microseconds (with still around 64 microseconds at 0.1% of line rate).[7] This increase in latency at low rates is not a real surprise:

- DPDK applications usually handle packets in bulks: such applications might wait up to some predefined time until a certain number of packets are available. These are then handled in bulk, as this is more efficient than handling packets one by one.

[7] Intel internal analysis. See Section 3 for the system under test's configuration details, and Section 4 for the test generators' configuraton details.

- Power saving techniques might also be implemented, either reducing the polling frequency when the load is low, or even maybe enabling C-states (power modes), allowing a server to switch into deeper C-states when the load is low. The Brocade 5600 vRouter supports parameters to tune sleep time (see power-profile, min-sleep, max-sleep parameters).

High latency is observed when the input rate is higher than 80% of the line rate: the system is then overloaded, and all hardware and software buffers and queues are full (e.g., including NIC RX buffer, RX descriptors, internal application buffers, inter-core ring buffers; TX descriptors, NIC TX buffer size).

No QoS was used in this test setup. The use of QoS would probably dramatically increase the latency (up to milliseconds), as many packets may be buffered in such a use case.

1.2.4 The Influence of the Number of Next Hops

A number of configurations were used with several fixed numbers of hosts/next hops to establish a realistic baseline performance.

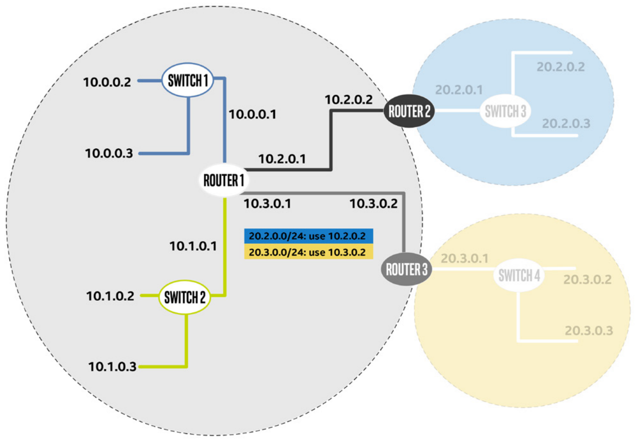


Figure 9. Next hops in network topology

As shown in Figure 9, there might be many next hops for a router: gateways (like Router 2 and Router 3), as well as some systems directly connected (though switches) to the router under test (like the systems connected through Switch 1 and Switch 2).

The performance when varying the number of next hops is shown in Figure 10. This test has been run five times for each of the data points, and error bars show the variability in performance. This variability is higher when very few next hops are used.

Increasing the number of hosts/next hops has an influence on the performance. 256-byte packets and higher can be handled by the system at line rate provided that less than 32k next hops are active per interface.[7]

From that perspective, it is important to note that a packet size of 256 bytes is probably lower than the average packet size of networks. So, being able to handle any packet size higher than 256 bytes is enough for most networks.

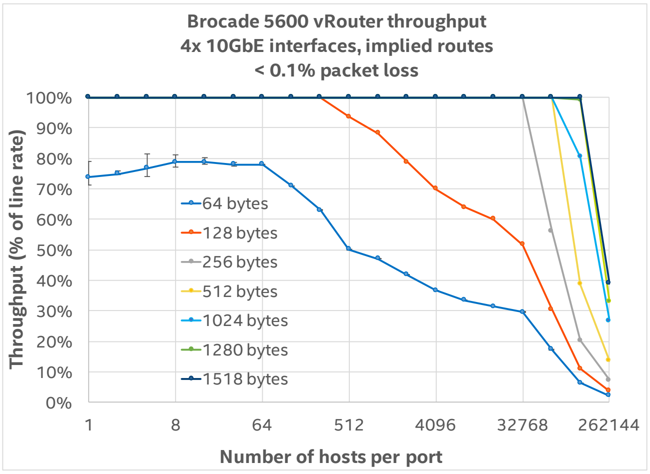


Figure 10. Influence of the number of next hops on throughput[7]

It is also probably quite unrealistic to have more than a few hundred systems connected to a router (through switches). Most networks probably do not use more than a few hundred next hops.

In summary, for realistic use cases (>= 256 byte packets, <= 1k active hosts per interface), the system can handle packets at line rate.

1.2.5 Influence of Number of Routes

Another important parameter that can influence router performance is the number of routes configured in the routing table, and consequently the number of systems that can be reached through the routing table (in Figure 11, systems in green or yellow networks, like 20.2.0.2).

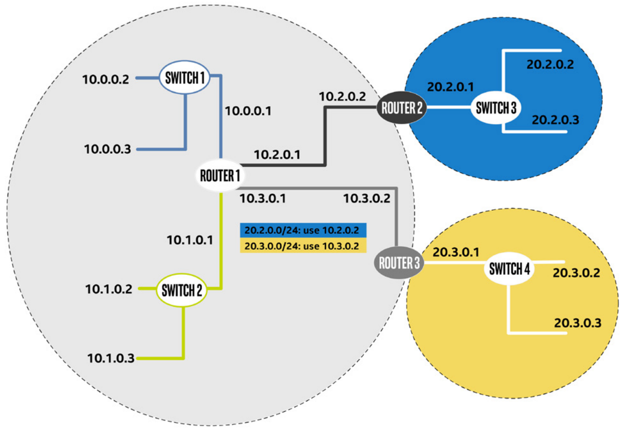


Figure 11. Routes in network topology

In addition to the number of routes, the netmask of the routes also influences performance.

Figure 12 shows the performance of the Brocade 5600 vRouter when increasing the number of /24 static routes. Up to 32k routes have been configured, i.e., 8k routes per interface. Performance is shown after running tests one time, while performance variability (as shown in Figure 10) still applies. Line rate is reached for all packet sizes except 64 bytes. In this case, performance is not influenced by the number of routes; the difference seen on the graph for 64 bytes packets fits within the variability identified in Figure 10.

Another approach would be to use dynamic routing protocols such as BGP to configure the routes, but this is out of the scope of this document.

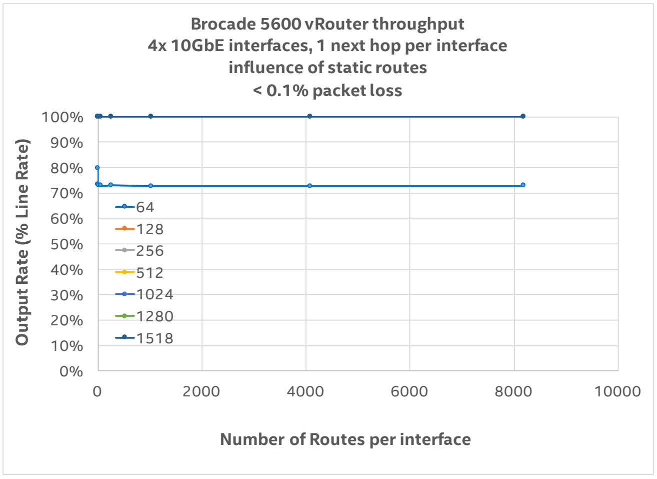


Figure 12. Influence of the number of routes on throughput[7]

We see in Figure 12 that for up to 32k routes, the number of routes has no influence on the performance.[7]

1.2.6 Two Brocade 5600 vRouter Instances on a Dual-Socket System

All test results so far were obtained using only one CPU socket. The goal of this additional characterization is to see how the Brocade 5600 vRouter performance scales with the number of sockets: two independent Brocade 5600 vRouter instances, each running on its own CPU socket, are being used (see Figure 13).

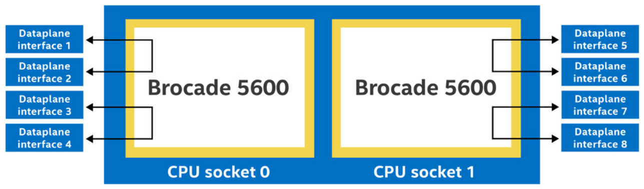


Figure 13. Two instance of vRouters, each on its own CPU socket

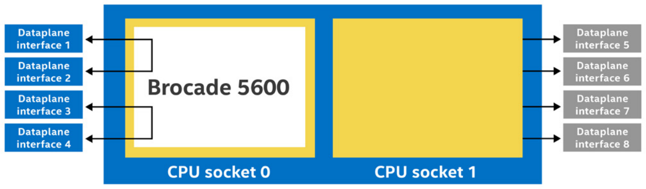


Figure 14. One vRouter instance

The comparison between the performance obtained with the router handling four interfaces using all cores from one CPU socket (Figure 14) is compared with the performance obtained when having two instances of the router, each handling four interfaces and using all cores from one CPU socket (Figure 13). Figure 15 shows that the performance scales very well with the number of sockets when using two VNFs, one on each socket.

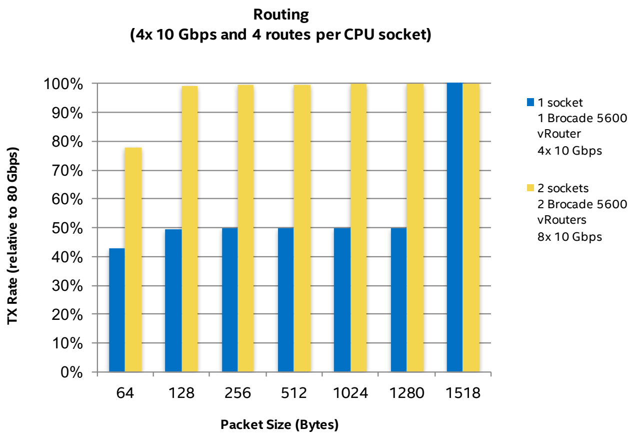


Figure 15. vRouter performance scaling[7]

1.3 Conclusion

The Brocade 5600 vRouter is an example of how an appliance-based function can be virtualized and achieve the benefits promised by NFV technology. To answer the performance question, this document provides a roadmap for interested parties to create their own test plan for determining whether the performance of the Brocade 5600 vRouter is suitable for their applications.

Configuration Specifications

This document explains how to properly configure the systems (both SUT and test systems) to run the Brocade 5600 vRouter characterization that was described in the pages above.

2 High-Level Overview

The Brocade 5600 vRouter runs in a virtual machine, using QEMU as the hypervisor and CentOS as the host operating system. PCI passthrough is being used, i.e., the control of the full physical device is given to the virtual machine; there is no virtual switch involved in the fast path (see Figure 3). Two different configurations can be used:

One Brocade 5600 vRouter instance using one CPU socket (Figure 16).

Two instances each using its own CPU socket (Figure 17).

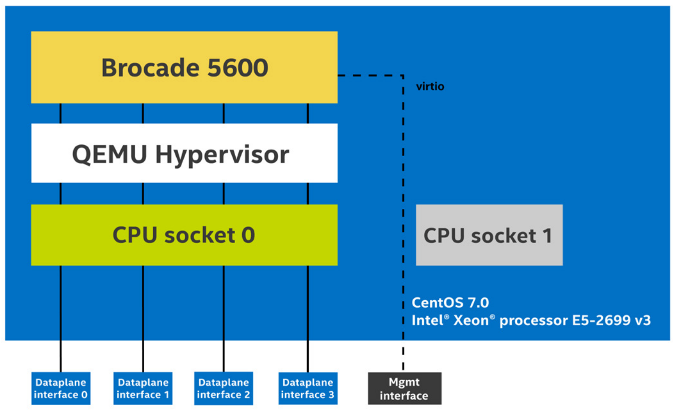


Figure 16. High level architecture – one instance on one socket

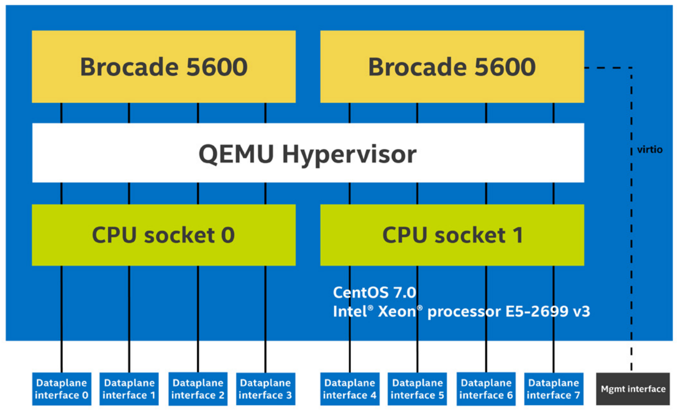


Figure 17. High level architecture: two instance, each on its own CPU socket

3 SUT Installation and Configuration

3.1 Host Configuration

3.1.1 Hardware and Software Details

ITEM	DESCRIPTION	NOTES
Platform	Intel® Server Board S2600WT Family	
Form factor	2U Rack Mountable	
Processor(s)	2x Intel® Xeon® CPU E5-2699 v3	46080KB L3 Cache per CPU, 18 cores per CPU (36 logical cores due to Hyper-threading).
Memory	32 GB RAM (8x 4 GB) per socket	Quad channel 2134 DDR4
BIOS	SE5C610.86B.01.01.0009.060120 151350	Hyper-threading enabled Hardware prefetching enabled COD disabled
Host OS	CentOS 7.1	Kernel version: 3.10.0-229.7.2.el7.x86_64
Hypervisor	QEMU	2.4.1
vRouter	Brocade 5600 vRouter v4.0R1	
Hugepages	8x 1 GB on host or 16x 1 GB on host 5x 1 GB in VM	16x used when using 8x 10 Gbps interfaces
DPDK	used	Used in Brocade 5600 vRouter; version unknown
NICs	8x Intel® 82599 10 Gigabit Ethernet Controller	2 dual-port PCIe gen-2 cards on socket 0 and 2 on socket 1.

3.1.2 Grub.cfg

DPDK-related CPU cores must be isolated through isolcpus in grub.cfg by making sure interrupts are handled by core 0.

```
     linux16 /vmlinuz-3.10.0-229.11.1.el7.
x86_64 root=/dev/mapper/centos-root ro
rd.lvm.lv=centos/swap vconsole.keymap=us
ipv6.disable=1 crashkernel=auto  vconsole.
font=latarcyrheb-sun16 rd.lvm.lv=centos/
root selinux=0 rhgb quiet LANG=en_US.UTF-8
intel_iommu=on iommu=pt noirqbalance intel_
pstate=disable intel_idle.max_cstate=0
processor.max_cstate=0 default_hugepagesz=1G
hugepagesz=1G hugepages=16 transparent_
hugepage=never
isolcpus=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,
51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,
68,69,70,71
```

3.1.3 QEMU

3.1.3.1 Dependencies

The following packages must be installed to be able to install and build QEMU.

```
sudo yum install net-tools
sudo yum install gcc
sudo yum install bzip2
sudo yum install zlib-devel
sudo yum install glib2-devel
sudo yum install gcc-c++
sudo yum install flex bison autoconf automake
libtool
sudo yum install numactl-devel numactl
sudo yum install pciutils
sudo yum install bridge-utils
```

3.1.3.2 QEMU Build

```
tar xvjf qemu-2.4.1.tar.bz2
cd qemu-2.4.1
./configure --disable-attr --enable-kvm
--enable-vhost-net --disable-docs --disable-
vnc-png --disable-vnc-jpeg --disable-sdl
--disable-curl --disable-curses --disable-
vnc-sasl --disable-vnc-tls --enable-numa
make
make install
```

3.1.4 Scripts

The following sections list the scripts used for the characterization. There is no guarantee that these scripts will run on software or hardware with different specifications than that which is listed in this document.

Scripts expect to have the Brocade 5600 vRouter image (`vyatta-kvm _ 4.0R1 _ amd64.img`) in user home directory (`/home/user` in this example). When using two VMs, the image of the second VM is called `vyatta-kvm _ 4.0R1 _ amd64 _ vm2.img`.

3.1.4.1 Scripts to Prevent Interrupts on DPDK Fast Path

Disable un-used services... (as root).

```
chkconfig abrt-ccpp off
chkconfig abrtd off
chkconfig acpid off
chkconfig atd off
chkconfig auditd off
chkconfig autofs off
chkconfig blk-availability off
chkconfig certmonger off
chkconfig cpuspeed off
chkconfig cups off
chkconfig haldaemon  off
chkconfig firewalld off
chkconfig ip6tables off
chkconfig iptables off
chkconfig irqbalance off
chkconfig kdump off
chkconfig ksmtuned off
chkconfig ksm off
chkconfig libvirt-guests off
chkconfig libvirtd off
chkconfig lvm2-monitor off
chkconfig mcelogd off
chkconfig mdmonitor off
chkconfig messagebus off
chkconfig netfs off
chkconfig nfs off
chkconfig nfslock off
chkconfig portreserve off
chkconfig postfix off
chkconfig rpcbind off
chkconfig rpcgssd off
chkconfig rpcidmapd off
chkconfig sysstat off

service abrt-ccpp stop
service abrtd stop
service acpid stop
service atd stop
service auditd stop
service autofs stop
service blk-availability stop
service certmonger stop
service cpuspeed stop
service cups stop
service haldaemon  stop
service ip6tables stop
service iptables stop
service irqbalance stop
service kdump stop
# Stop firewall - preventing vnc to qemu
service firewalld stop
service ksmtuned stop
service ksm stop
service libvirt-guests stop
service libvirtd stop
service lvm2-monitor off
service mcelogd stop
service mdmonitor stop
service messagebus stop
service netfs stop
service nfs stop
service nfslock stop
service portreserve stop
service postfix stop
service rpcbind stop
service rpcgssd stop
service rpcidmapd stop
service sysstat stop

rmmod bluetooth
rmmod rfkill
rmmod cpufreq _ stats
rmmod ip6table _ filter
rmmod ip6 _ tables
rmmod ebtable _ nat
rmmod ebtables
rmmod nf _ conntrack _ ipv4
rmmod nf _ defrag _ ipv4
rmmod xt _ state
rmmod nf _ conntrack
rmmod ipt _ REJECT
rmmod xt _ CHECKSUM
rmmod iptable _ mangle
rmmod iptable _ filter
rmmod ip _ tables
rmmod stp
rmmod llc
rmmod ipv6
rmmod dm _ mirror
rmmod dm _ region _ hash
rmmod dm _ log
rmmod dm _ mod
rmmod vhost _ net
rmmod macvtap
rmmod macvlan
rmmod vhost
rmmod tun
rmmod iTCO _ wdt
rmmod iTCO _ vendor _ support
rmmod microcode
rmmod pcspkr
rmmod lpc _ ich
rmmod mfd_core
rmmod i2c_algo_bit
rmmod dca
rmmod ptp
rmmod pps_core
rmmod mdio
rmmod sg
rmmod i2c_i801
rmmod i2c_core
rmmod wmi
rmmod ext4
rmmod jbd2
rmmod mbcache
rmmod sd_mod
rmmod crc_t10dif
rmmod crct10dif_common
rmmod ahci
rmmod libahci
rmmod isci
rmmod libsas
rmmod scsi_transport_sas

echo 0 > /proc/sys/kernel/nmi_watchdog
echo 0 > /proc/sys/kernel/numa_balancing
echo 1 > /sys/bus/workqueue/devices/writeback/cpumask
sysctl vm/stat_interval=1000000

killall sftp-server
killall udevd

for i in `pgrep rcu[^c]` ; do taskset -pc 0 $i ; done

./set_irq_affinity enp3s0f0
```

Where enp3s0f0 is the management interface and set_irq_affinity is defined by:

```
device=$1
if [ $device = "" ] ; then
  echo "Please select which interface to use"
  exit
fi

i=0
while [ 1 ]; do
  irq=`cat /proc/interrupts | grep -i $device-TxRx-$i"$"  | cut  -d:  -f1 | sed "s/ //g"`
  if [ -n  "$irq" ]; then
    printf "1 > /proc/irq/%d/smp_affinity\n" $irq
    printf 1 > /proc/irq/$irq/smp_affinity
    i=$(($i+1))
  else
    exit
  fi
done
```

3.1.4.2 Hugepages, vfio...

```
# Create and mount hugepages
sudo mkdir -p /mnt/huge
sudo mount -t hugetlbfs nodev /mnt/huge

# Create tap device
ip tuntap add tap0 mode tap
ip tuntap add tap1 mode tap
ifconfig tap0 up
ifconfig tap1 up
```

```
# Create bridge for mgmt. interface
brctl addbr br0
brctl addif br0 tap0
brctl addif br0 tap1

# Load and bind vfio driver
modprobe vfio-pci
echo "8086 10fb" > /sys/bus/pci/drivers/vfio-pci/new _ id
echo 0000:05:00.0 > /sys/bus/pci/devices/0000\:05\:00.0/driver/unbind
echo 0000:05:00.0 > /sys/bus/pci/drivers/vfio-pci/bind
echo "8086 10fb" > /sys/bus/pci/drivers/vfio-pci/new _ id
echo 0000:05:00.1 > /sys/bus/pci/devices/0000\:05\:00.1/driver/unbind
echo 0000:05:00.1 > /sys/bus/pci/drivers/vfio-pci/bind
echo "8086 10fb" > /sys/bus/pci/drivers/vfio-pci/new _ id
echo 0000:07:00.0 > /sys/bus/pci/devices/0000\:07\:00.0/driver/unbind
echo 0000:07:00.0 > /sys/bus/pci/drivers/vfio-pci/bind
echo "8086 10fb" > /sys/bus/pci/drivers/vfio-pci/new _ id
echo 0000:07:00.1 > /sys/bus/pci/devices/0000\:07\:00.1/driver/unbind
echo 0000:07:00.1 > /sys/bus/pci/drivers/vfio-pci/bind
echo "8086 10fb" > /sys/bus/pci/drivers/vfio-pci/new _ id
echo 0000:81:00.0 > /sys/bus/pci/devices/0000\:81\:00.0/driver/unbind
echo 0000:81:00.0 > /sys/bus/pci/drivers/vfio-pci/bind
echo "8086 10fb" > /sys/bus/pci/drivers/vfio-pci/new _ id
echo 0000:81:00.1 > /sys/bus/pci/devices/0000\:81\:00.1/driver/unbind
echo 0000:81:00.1 > /sys/bus/pci/drivers/vfio-pci/bind
echo "8086 10fb" > /sys/bus/pci/drivers/vfio-pci/new _ id
echo 0000:83:00.0 > /sys/bus/pci/devices/0000\:83\:00.0/driver/unbind
echo 0000:83:00.0 > /sys/bus/pci/drivers/vfio-pci/bind
echo "8086 10fb" > /sys/bus/pci/drivers/vfio-pci/new _ id
echo 0000:83:00.1 > /sys/bus/pci/devices/0000\:83\:00.1/driver/unbind
echo 0000:83:00.1 > /sys/bus/pci/drivers/vfio-pci/bind

service NetworkManager stop

vi /etc/sysconfig/network-scripts/ifcfg-br0
    DEVICE=br0
    #BOOTPROTO=dhcp
    BOOTPROTO=static
    IPADDR=192.168.1.142
    NETMASK=255.255.255.0
    GATEWAY=192.168.1.240
    ONBOOT=yes
    TYPE=Bridge

vi /etc/sysconfig/network-scripts/ifcfg-enp3s0f0
# Generated by dracut initrd
    NAME="enp3s0f0"
    DEVICE="enp3s0f0"
    ONBOOT=yes
    UUID="a4d56fab-015e-458a-bb40-6a08cb8cf8d9"
    TYPE=Ethernet
    BRIDGE=br0
service network restart
```

3.1.4.3 QEMU Startup Script

See 3.1.4.4 for information on `start _ vm.py` script used in this section.

3.1.4.3.1 One VM on Socket 0

```
python start _ vm.py -name centos -enable-kvm \
        -cpu host \
        -m 8192 \
        -object memory-backend-file,prealloc=yes,mem-path=/mnt/huge,size=8192M,id=ram-
node0,host-nodes=0,policy=bind \
        -numa node,nodeid=0,cpus=0-17,memdev=ram-node0 \
        -uuid 8da5a07e-3b51-4be1-9f24-b8061c1dc271 -boot order=d   \
        -drive file=/home/user/vyatta-kvm _ 4.0R1 _ amd64.img,if=none,id=drive-ide0-0-1,
format=raw \
        -device ide-hd,bus=ide.0,unit=0,drive=drive-ide0-0-1,id=ide0-0-1 \
        -net nic,model=e1000,macaddr=52:54:00:90:f2:a2 \
        -net tap,ifname=tap0,script=no,downscript=no,vhost=on \
        -vnc 192.168.1.142:2 \
        -device vfio-pci,host=05:00.0,id=hostdev0,bus=pci.0,addr=0x3 \
        -device vfio-pci,host=05:00.1,id=hostdev1,bus=pci.0,addr=0x4 \
        -device vfio-pci,host=07:00.0,id=hostdev2,bus=pci.0,addr=0x5 \
        -device vfio-pci,host=07:00.1,id=hostdev3,bus=pci.0,addr=0x6 \
        -device cirrus-vga,id=video0,bus=pci.0,addr=0x7 \
        -daemonize
```

3.1.4.3.2 Two VMs - One VM on Each CPU Socket

```
python start _ vm.py -name centos -enable-kvm \
        -cpu host \
        -m 8192 \
        -object memory-backend-file,prealloc=yes,mem-path=/mnt/huge,size=8192M,id=ram-
node0,host-nodes=0,policy=bind \
        -numa node,nodeid=0,cpus=0-17,memdev=ram-node0 \
        -uuid 8da5a07e-3b51-4be1-9f24-b8061c1dc271 -boot order=d   \
        -drive file=/home/user/vyatta-kvm _ 4.0R1 _ amd64.img,if=none,id=drive-ide0-0-1,
format=raw \
        -device ide-hd,bus=ide.0,unit=0,drive=drive-ide0-0-1,id=ide0-0-1 \
        -net nic,model=e1000,macaddr=52:54:00:90:f2:a2 \
        -net tap,ifname=tap0,script=no,downscript=no,vhost=on \
        -vnc 192.168.1.142:2 \
        -device vfio-pci,host=05:00.0,id=hostdev0,bus=pci.0,addr=0x3 \
        -device vfio-pci,host=05:00.1,id=hostdev1,bus=pci.0,addr=0x4 \
        -device vfio-pci,host=07:00.0,id=hostdev2,bus=pci.0,addr=0x5 \
        -device vfio-pci,host=07:00.1,id=hostdev3,bus=pci.0,addr=0x6 \
        -device cirrus-vga,id=video0,bus=pci.0,addr=0x7 \
        -daemonize

python start _ vm2.py -name centos -enable-kvm \
        -cpu host \
        -m 8192 \
        -object memory-backend-file,prealloc=yes,mem-path=/mnt/huge,size=8192M,id=ram-
node1,host-nodes=1,policy=bind \
        -numa node,nodeid=0,cpus=0-17,memdev=ram-node1 \
        -uuid 8da5a07e-3b51-4be1-9f24-b8061c1dc271 -boot order=d   \
        -drive file=/home/user/vyatta-kvm _ 4.0R1 _ amd64 _ vm2.img,if=none,id=drive-ide0-0-
1,format=raw \
        -device ide-hd,bus=ide.0,unit=0,drive=drive-ide0-0-1,id=ide0-0-1 \
        -net nic,model=e1000,macaddr=52:54:00:90:f2:a3 \
        -net tap,ifname=tap1,script=no,downscript=no,vhost=on \
        -vnc 192.168.1.142:3 \
        -device vfio-pci,host=81:00.0,id=hostdev0,bus=pci.0,addr=0x3 \
        -device vfio-pci,host=81:00.1,id=hostdev1,bus=pci.0,addr=0x4 \
        -device vfio-pci,host=83:00.0,id=hostdev2,bus=pci.0,addr=0x5 \
        -device vfio-pci,host=83:00.1,id=hostdev3,bus=pci.0,addr=0x6 \
        -device cirrus-vga,id=video0,bus=pci.0,addr=0x7 \
        -daemonize
```

3.1.4.4 Scripts for Pinning Virtual CPUs to Physical Cores

On the host system, every QEMU thread must be pinned to a different CPU core through taskset.

`start _ vm.py` script provided by prox (in helper-scripts) can be used to launch a VM and pin virtualized cores to physical cores. The right pinning must be provided in `vm-cores.py` script.

This script uses the following syntax when Hyperthread is disabled:

`cores = [[0], [1], [2], [3], [4], [5], [6], [7], [8], [9]]` to map virtual core 0 to 9 to physical core 0 to 9

3.1.4.4.1 One VM on Socket 0

Configure the script to run on 18 cores

```
cores = [[1],[2],[3],[4],[5],[6],[7],[8],[9],[10],[11],[12],[13],[14],[15],[16],[17],[0]]
```

3.1.4.4.2 Two VMs - One VM on Each CPU Socket

Configure the script to run on 18 cores in `vm-cores.py`

```
cores = [[1],[2],[3],[4],[5],[6],[7],[8],[9],[10],[11],[12],[13],[14],[15],[16],[17],[0]]
```

Copy the script to start_vm2.py. Modify it to get core information from `vm2-cores.py` and configure the `vm2-cores.py` script to run on 18 cores of socket 1.

```
cores = [[18],[19],[20],[21],[22],[23],[24],[25],[26],[27],[28],[29],[30],[31],[32],[33],[34],[35]]
```

3.2 vRouter Configuration

3.2.1 Login and Password

Login=vyatta

Password=vyatta

3.2.2 Set Root Password

```
configure
set system login user root authentication plaintext-password 123456
commit
save
exit
```

3.2.3 Set vRouter Management Interface IP address

```
set interfaces dataplane dp0s2 address 192.168.1.210/24
set system static-host-mapping host-name vyatta inet 192.168.1.210
```

3.2.4 Enable ssh Access + http

```
configure
set service ssh
set service ssh allow-root
set service http
commit
save
exit
```

3.2.5 Key Manipulation

Copy public key from prox management system (from which prox characterization script will be started) to `/home/vyatta/pk.pub`; then have the vRouter properly load the keys. Note: copying them manually in `.ssh` will result in those changes being lost after reboot.

```
configure
loadkey root /home/vyatta/pk.pub
commit
save
exit
```

3.2.6 Set Dataplane IP Address

This is only one example. IP addresses and routes vary per use case.

```
set interfaces dataplane dp0s3 address 64.0.0.240/24
set interfaces dataplane dp0s4 address 65.0.0.240/24
set interfaces dataplane dp0s5 address 66.0.0.240/24
set interfaces dataplane dp0s6 address 67.0.0.240/24
```

3.2.7 Create Routes

```
set protocols static route 1.0.0.0/24 next-hop 64.0.0.1
set protocols static route 9.0.0.0/24 next-hop 65.0.0.1
set protocols static route 17.0.0.0/24 next-hop 66.0.0.1
set protocols static route 25.0.0.0/24 next-hop 67.0.0.1
```

3.2.8 Example Config File

The commands should result in a config file similar to this one:

```
interfaces {
        dataplane dp0s2 {
                address 192.168.1.210/24
        }
        dataplane dp0s3 {
                address 64.0.0.240/24
        }
        dataplane dp0s4 {
                address 65.0.0.240/24
        }
        dataplane dp0s5 {
                address 66.0.0.240/24
        }
        dataplane dp0s6 {
                address 67.0.0.240/24
        }
        loopback lo
}
protocols {
        static {
                route 1.0.0.0/24 {
                        next-hop 64.0.0.1
                }
                route 9.0.0.0/24 {
                        next-hop 65.0.0.1
                }
                route 17.0.0.0/24 {
                        next-hop 66.0.0.1
                }

                route 25.0.0.0/24 {
                        next-hop 67.0.0.1
                }
        }
}
service {
        https
        ssh
}
system {
        acm {
                enable
                operational-ruleset {
                        rule 9985 {
                                action allow
                                command /show/tech-support/brief/
                                group vyattaop
                        }
                        rule 9986 {
                                command /show/tech-support/brief
                                group vyattaop
                        }
                        rule 9987 {
                                command /show/tech-support
                                group vyattaop
                        }
                        rule 9988 {
                                command /show/configuration
                                group vyattaop
```

```
          }
          rule 9989 {
                  action allow
                  command "/clear/*"
                  group vyattaop
          }
          rule 9990 {
                  action allow
                  command "/show/*"
                  group vyattaop
          }
          rule 9991 {
                  action allow
                  command "/monitor/*"
                  group vyattaop
          }
          rule 9992 {
                  action allow
                  command "/ping/*"
                  group vyattaop
          }
          rule 9993 {
                  action allow
                  command "/reset/*"
                  group vyattaop
          }
          rule 9994 {
                  action allow
                  command "/release/*"
                  group vyattaop
          }
          rule 9995 {
                  action allow
                  command "/renew/*"
                  group vyattaop
          }
          rule 9996 {
                  action allow
                  command "/telnet/*"
                  group vyattaop
         }
          rule 9997 {
                  action allow
                  command "/traceroute/*"
                  group vyattaop
          }
          rule 9998 {
                  action allow
                  command "/update/*"
                  group vyattaop
          }
          rule 9999 {
                  command "*"
                  group vyattaop
          }
     }
     ruleset {
          rule 9999 {
                  action allow
                  group vyattacfg
                  operation "*"
                  path "*"
          }
     }
```

```
        }
        config-management {
                commit-revisions 20
        }
        console {
                device ttyS0
        }
        domain-name mcplab.net
        host-name Vyatta-5600
        login {
                user root {
                     authentication {
                             encrypted-password $1$MGW0BV.5$vTG5Jtx6wPPxUGJmqalFH1
                             public-keys root@vRouter.mcplab.net {
                                     key
AAAAB3NzaC1yc2EAAAADAQABAAABAQC5Mp83HPbAKauvCejhYAINjSB/CFv/6mBFwg+DmshLORPKtfBM+zvBPdeOL1GXZ
2sXJxUIz1HYWJ0ePsnsM05DfbdVfDN8D6s5uUgIsLM4wXx0jj17wcynFE2FBHq0FWab4fnj2ZqOLY9Z4UA63TFvBTdUfAz
xP8M/sSQGsR2nWTZ2300yH9SK8v4khClAFlQRt4Qp06ltMjo6QzHAUH3z8BGRkoF0LpJ9mayAN0UZ6QcE24kZJUQMFLFrJi
DfxzUwWDSdL2U0s5HDkHXyFFlJ6x/gTFjee7hl8njlgIN7gG/uT9OWpdLI/jlUg3VgR4h8hOdsPBcGSbhMpAUo39P3
                                     type ssh-rsa
                             }
                     }
                }
                user vyatta {
                     authentication {
                             encrypted-password $1$MhxqymQQ$hpLZRkWd0hI3f5UQ0Z0ZO.
                             public-keys user@vRouter.mcplab.net {
                                     key
AAAAB3NzaC1yc2EAAAADAQABAAABAQC5Mp83HPbAKauvCejhYAINjSB/CFv/6mBFwg+DmshLORPKtfBM+zvBPdeOL1GXZ2
sXJxUIz1HYWJ0ePsnsM05DfbdVfD6s5uUgIsLM4wXx0jj17wcynFE2FBHq0FWab4fnj2ZqOLY9Z4UA63TFvBTdUfAzxP8M/sS
QGsR2nWTZ2300yH9SK8v4khClAFlQRt4Qp06ltMjo6QzHAUH3z8BGRkoF0LpJ9mayAN0UZ6QcE24kZJUQMFLFrJiDfxzUwWD
DSdL2U0s5HDkHXyFFlJ6x/gTFjee7hl8njlgIN7gG/uT9OWpdLI/jlUg3VgR4h8hOdsPBcGSbhMpAUo39P3
                                     type ssh-rsa
                             }
                     }
                     level admin
                }
        }
        static-host-mapping {
                host-name Vyatta-5600 {
                     inet 192.168.1.96
                }
        }
        syslog {
                global {
                     facility all {
                             level warning
                     }
                }
        }
}


/* Warning: Do not remove the following line. */
/* === vyatta-config-version: "config-management@1:dhcp-relay@2:pim@1:qos@2:routing@5:sflow@1:system
@13:twamp@1:vlan@1:vplane@2:vrrp@1:vrrp@2:webgui@1" === */
/* Release version: 3.5R5 */
```

3.2.8.1 VM1 Configuration

The configuration of the interfaces and routes of the first VM, with four routes and four next hops:

```
interfaces {
        dataplane dp0s2 {
                address 192.168.1.210/24
        }
        dataplane dp0s3 {
                address 64.0.0.240/24
        }
        dataplane dp0s4 {
                address 65.0.0.240/24
        }
        dataplane dp0s5 {
                address 66.0.0.240/24
        }
        dataplane dp0s6 {
                address 67.0.0.240/24
        }
        loopback lo
}
protocols {
        static {
                route 1.0.0.0/24 {
                        next-hop 64.0.0.1
                }
                route 9.0.0.0/24 {
                        next-hop 65.0.0.1
                }
                route 17.0.0.0/24 {
                        next-hop 66.0.0.1
                }
                route 25.0.0.0/24 {
                        next-hop 67.0.0.1
                }
        }
}
```

3.2.8.2 VM2 Configuration

The configuration of the interfaces and routes of the second VM, with four routes and four next hops (use case 1).

```
interfaces {
        dataplane dp0s2 {
                address 192.168.1.211/24
        }
        dataplane dp0s3 {
                address 68.0.0.240/24
        }
        dataplane dp0s4 {
                address 69.0.0.240/24
        }
        dataplane dp0s5 {
                address 70.0.0.240/24
        }
        dataplane dp0s6 {
                address 71.0.0.240/24
        }
        loopback lo
}
protocols {
        static {
                route 33.0.0.0/24 {
                        next-hop 68.0.0.1
                }
                route 41.0.0.0/24 {
                        next-hop 69.0.0.1
                }
                route 49.0.0.0/24 {
                        next-hop 70.0.0.1
                }
                route 57.0.0.0/24 {
                        next-hop 71.0.0.1
                }
        }
}
```

3.2.9 Example Script for Creating Brocade 5600 vRouter Configuration Files

The characterization tool (see section 4.4) uses many different configuration files. It provides a vRouter-agnostic simple script for showing the interface addresses and routes used during the characterization. That information can then be used to create configuration files. Or the script can be modified to directly create the config file using the correct syntax for the vRouter under test. An example of such a modified script is given here below for Brocade 5600 vRouter. The config*.boot must be copied in /config/prox directory on the Brocade 5600 vRouter.

3.2.9.1 One VM on Socket 0 – Four Ports per VM

```
use strict;
my $max_nb_routes = 8192;
my $max_nb_next_hops = 1024;
my $max_nb_interfaces = 4;
my $nb_next_hops = 1;
my ($interface, $a1, $a2, $a3, $a4, $fh, $output_route);

# Create bricade configuration for use case 0 and 1
while ($nb_next_hops <= $max_nb_next_hops) {
        my $nb_routes_per_interface = $nb_next_hops;
        while ($nb_routes_per_interface <= $max_nb_routes) {
                my $config = "tmp.".$nb_routes_per_interface."_".$nb_next_hops.".boot";
                open($fh, '>', $config) or die "Could not open file '$config' $!";
                print $fh "interfaces {\n";
                print $fh "\tdataplane dp0s2 {\n";
                print $fh "\t\taddress 192.168.1.96/24\n";
                print $fh "\t}\n";
                for (my $i = 0; $i < $max_nb_interfaces; $i++) {
                        print $fh "\tdataplane dp0s".($i+3)." {\n";
                        print $fh "\t\taddress ".($i+64).".0.0.240/24\n";
                        print $fh "\t}\n";
                }
                print $fh "\tloopback lo\n}\n";
                print $fh "protocols {\n";
                print $fh "\tstatic {\n";

                for (my $route_nb = 0; $route_nb < $nb_routes_per_interface; $route_
nb++) {
                        for ($interface = 0; $interface < $max_nb_interfaces; $interface++) {
                                $a1 = $interface * 8 + 1 + (($route_nb & 1) << 2) + ($route_
nb & 2);
                                $a2 = (($route_nb & 4) << 5) + (($route_nb & 8) << 1) +
(($route_nb & 0x10) >> 1) + (($route_nb & 0x20) >> 4) + (($route_nb & 0x40) >> 6);
                                $a3 = (($route_nb & 0x80)) + (($route_nb & 0x100) >> 2)
+ (($route_nb & 0x200) >> 5) + (($route_nb & 0x400) >> 7) + (($route_nb & 0x800) >> 10) +
(($route_nb & 0x1000) >> 12);
                                $a4 = 0;
                                print $fh "\t\troute $a1.$a2.$a3.$a4"."/24 {\n";
                                print $fh "\t\t\tnext-hop ".($interface+64).".0.".(($route_nb %
$nb_next_hops) >> 7).".".(1 + (($route_nb % $nb_next_hops) & 0x7f)) ."\n";
                                print $fh "\t\t}\n";
                        }
                }
                print $fh "\t}\n";
                print $fh "}\n";
                close $fh;
                my $output _ file = "config.".$nb _ routes _ per _ interface." _ ".$nb _ next _
hops.".boot";
                my $cmd = "cat $config config.base > $output _ file";
                system($cmd);
                $nb _ routes _ per _ interface = $nb _ routes _ per _ interface * 2;
        }
        $nb _ next _ hops = $nb _ next _ hops * 2;
}

# Create routes configuration for use case 2

my $config = "tmp.1 _ 1 _ 2.boot";
open($fh, '>', $config) or die "Could not open file '$config' $!";

print $fh "interfaces {\n";
print $fh "\tdataplane dp0s2 {\n";
print $fh "\t\taddress 192.168.1.96/24\n";
print $fh "\t}\n";
for (my $i = 0; $i < $max _ nb _ interfaces; $i++) {
        print $fh "\tdataplane dp0s".($i+3)." {\n";
        print $fh "\t\taddress ".($i*8+1).".0.0.240/5\n";
        print $fh "\t}\n";
}
print $fh "\tloopback lo\n}\n";
print $fh "protocols {\n";
print $fh "\tstatic {\n";

for (my $i = 0; $i < $max _ nb _ interfaces; $i++) {
        $a1 = $i + 64 ;
        $a2 = 0;
        $a3 = 0;
        $a4 = 0;
        print $fh "\t\troute $a1.$a2.$a3.$a4/24 {\n";
        print $fh "\t\t\tnext-hop ".($interface * 8 + 1).".0.0.1\n";
        print $fh "\t\t}\n";
}
print $fh "\t}\n";
print $fh "}\n";
close $fh;
my $output _ file = "config.1 _ 1 _ 2.boot";
my $cmd = "cat $config config.base > $output _ file";
system($cmd);
```

In this script, `config.base` is a copy of `config.boot` as configured earlier in this document, where the protocol and interface fields have been deleted. IP address of management interface is hard-coded in the script (192.168.1.96 in this case).

3.2.9.2 Two VMs – One VM per CPU Socket, Four Ports per VM

This configuration has been used in only a limited set of tests. Hence the configuration files were modified by hand in the second VM.

- The management interface has been changed in the second VM.
- The datapath interfaces, IP addresses, and routes are changed as well, following the same logic as in the one VM configuration.

4 Test Generators' Installation and Configuration

4.1 Hardware and Software Details

ITEM	DESCRIPTION	NOTES
Platform	Intel® Server Board S2600WT Family	
Form factor	2U Rack Mountable	
Processor(s)	2x Intel® Xeon® CPU E5-2699 v3	46080KB L3 Cache per CPU, 18 cores per CPU (36 logical cores due to Hyper-threading).
Memory	32 GB RAM (8x 4 GB) per socket	Quad channel 2134 DDR4
BIOS	SE5C610.86B.01.01 0009.060120151350	Hyper-threading enabled Hardware prefetching enabled COD disabled

ITEM	DESCRIPTION	NOTES
Host OS	CentOS 7.1	Kernel version: 3.10.0-229.7.2.el7.x86_64
Hugepages	8x 1 GB	
PROX	0.31	http://github.com/nvf-crucio/prox
DPDK	2.2.0	
NICs	8x Intel® 82599 10 Gigabit Ethernet Controller	

4.1.1 Grub.cfg

DPDK-related CPU cores must be isolated through isolcpus in grub.cfg, by making sure interrupts are handled by core 0.

```
     linux16 /vmlinuz-3.10.0-229.11.1.el7.
x86 _ 64 root=/dev/mapper/centos-root ro
rd.lvm.lv=centos/swap vconsole.keymap=us
ipv6.disable=1 crashkernel=auto  vconsole.
font=latarcyrheb-sun16 rd.lvm.lv=centos/
root selinux=0 rhgb quiet LANG=en _ US.UTF-8
intel _ iommu=on iommu=pt noirqbalance intel _
pstate=disable intel _ idle.max _ cstate=0
processor.max _ cstate=0 default _ hugepagesz=1G
hugepagesz=1G hugepages=16 transparent _
hugepage=never
isolcpus=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,
51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,
68,69,70,71
```

4.1.2 Scripts to Prevent interrupts on DPDK Fast Path

Same script as in 3.1.4.1 is run to decrease interrupts.

4.2 Test Setup Details

The Brocade 5600 vRouter is characterized using test generators, and these test generators generate IP traffic towards four or eight 10 Gbps interfaces and measure the traffic coming from those interfaces. These test generators can be Ixia (or Spirent), or COTS servers running DPDK-based traffic-generation applications (pktgen or prox).

For automation purposes, prox (https://01.org/intel-data-plane-performance-demonstrators/prox-overview) has been used to generate the traffic and to measure the throughput and latency from the vRouter. Ixia equipment has been used as well to confirm some key data results.

Prox contains configuration files to be used for four- and eight-interface vRouters. It also contains a set of characterization scripts.

4.3 Test Parameters

The characterization studies the influence of the following parameters on the throughput and latency:

- Packet size
- Number of next hops
- Number of routes

In order to run the characterization, three "use cases" have been defined.

4.3.1 Use Case 0

In use case 0, the test generator increases the load of the system from 0% to 100% of the line rate on all interfaces. Steps of 0.01% are used from 0% to 0.1%; steps of 0.1% are used from 0.1% to 1%, and steps of 1% are used for each additional test. Each use case 0 run produces 120 data points. The throughput and latency generated by the SUT (vRouter) is measured. The resulting file produced by the characterization scripts contains the incoming and outgoing load as well as latency for those increasing loads. This can be easily plotted to show the influence of the load on dropped packets and on latency. Use case 0 is usually run for multiple packet sizes and traffic profiles (see below).

4.3.2 Use Case 1

Use case 1 measures 0 or 0.01% (configurable) packet loss, for a fixed number of routes and next hops. Each use case 1 run produces 1 data point: the maximum throughput rate for which 0 (or 0.01%) packets are lost. This use case is usually run for a varying number of next hops, number of routes, packet sizes, and traffic profiles. This use case mainly studies the influence of the number of routes and next hops on performance.

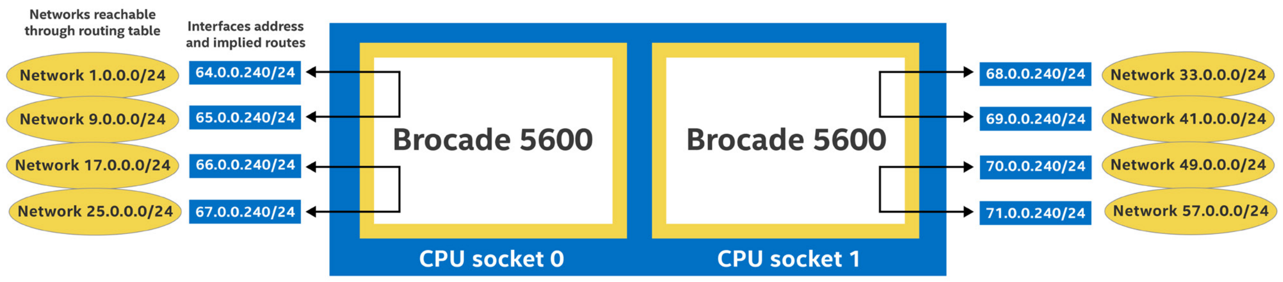


Figure 18. Network configuration

4.3.3 Use Case 2

Use case 2 focuses on the influence of the number of next hops. In this use case, no (or a very small number) of routes are used; packets are forwarded using the interface implicit route (i.e., the interface has an IP address and a prefix a.0.0.0/24, and this interface is used to forward packets a.0.0.b). Like use case 1, use case 2 measures maximum throughput using 0 or 0.01% packet loss. It is then run multiple times using a varying number of next hops.

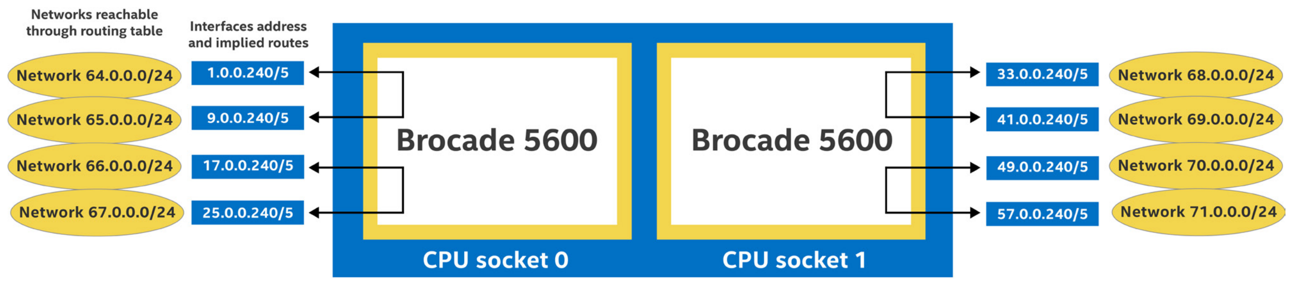


Figure 19. Network configuration

4.3.4 Traffic Profiles

Six different traffic profiles have been defined. Some of these profiles only make sense to use when testing the dual-socket configuration.

For the test results presented in this document, characterization is performed using the first traffic profile (traffic profile 0, i.e., for each interface, the traffic is sent to a unique other interface on the same CPU socket—from 1 to 2 and 2 to 1; from 3 to 4 and 4 to 3...)

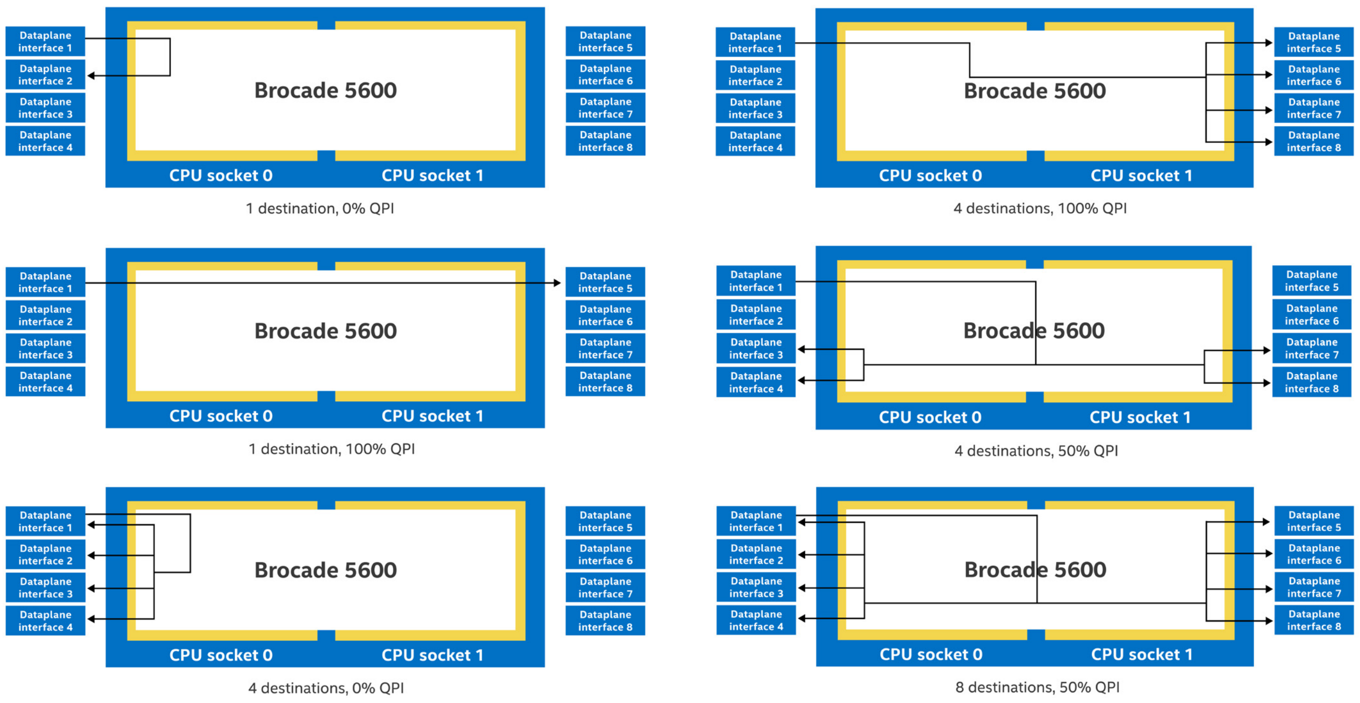


Figure 20. Traffic profiles

For each traffic profile, traffic is received on all interfaces and sent towards some interfaces following a predefined pattern.

- In traffic profile 0, traffic from interface 1 is sent to interface 2, and from interface 2 towards interface 1; from interface 3 towards 4 and from 4 towards 3.
- In traffic profile 1 (dual socket only), traffic is also sent towards only one outgoing interface for each incoming interface, but the outgoing interface is always on the other socket compared to the incoming interface.
- In traffic profile 2, each packet from an incoming interface can be forwarded to all outgoing interfaces on the same socket.
- In traffic profile 3 (dual socket only), each packet from an incoming interface can be forwarded to all outgoing interfaces on the other socket.
- In traffic profile 4 (dual socket only), all packets from an incoming interface are forwarded to two interfaces on socket 0 and two interfaces on socket 1 (50% of the packets cross QPI).
- In traffic profile 5 (dual socket only), each packet from an incoming interface can be forwarded to any outgoing interface (general vRouter case).

A traffic profile test is obtained by properly configuring the destination IP addresses of the packets being generated. Routes and next hop configurations are the same for all traffic profiles.

4 4 Characterization Scripts

The characterization is performed in two phases: first a configuration file listing the tests to execute is created. Then the tests are performed based on the configuration file.

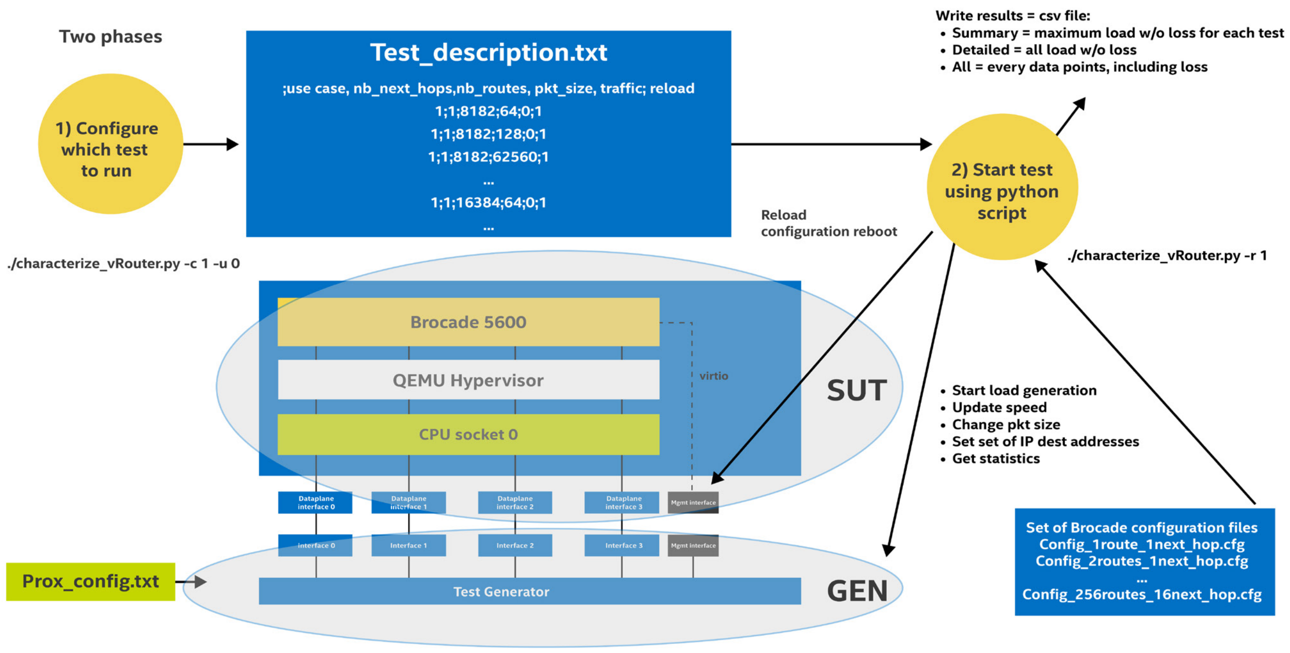

Two phases
1) Configure which test to run
Test_description.txt
;use case, nb_next_hops,nb_routes, pkt_size, traffic; reload
1;1;8182;64;0;1
1;1;8182;128;0;1
1;1;8182;62560;1
...
1;1;16384;64;0;1
...
Write results = csv file:
• Summary = maximum load w/o loss for each test
• Detailed = all load w/o loss
• All = every data points, including loss
2) Start test using python script
Reload configuration reboot
./characterize_vRouter.py -c 1 -u 0
./characterize_vRouter.py -r 1
Brocade 5600
QEMU Hypervisor
CPU socket 0
virtio
SUT
Dataplane interface 0
Dataplane interface 1
Dataplane interface 2
Dataplane interface 3
Mgmt interface
interface 0
Interface 1
Interface 2
Interface 3
Mgmt interface
Prox_config.txt
Test Generator
GEN
• Start load generation
• Update speed
• Change pkt size
• Set set of IP dest addresses
• Get statistics
Set of Brocade configuration files
Config_1route_1next_hop.cfg
Config_2routes_1next_hop.cfg
...
Config_256routes_16next_hop.cfg

Figure 21. Characterization overview

The characterization configuration file (test_description.txt) has the following format:

```
Use case; next _ hops; routes; pkt _ size; traffic; reload
```

Default characterization configurations can be built using the following command:

```
./characterize _ vRouter.py -c 1 -u x # where x is the use case number
```

In order to run the characterization, the vRouter must be configured accordingly. To accomplish this, a set of vRouter configurations can be produced using another script: create_interfaces_and_routes.pl. This script creates four sets of files: two sets for use case 2, and two sets for use case 0 or 1 (which use the same configurations). Each set contains:

- `Interface.txt`: contains the IP addresses and prefixes of the DPDK interfaces.
- `Route.x.y.txt`: contains the routing table for a configuration using x routes and y next hops.

These configurations files cannot be used as such by the vRouter. They are created using a vRouter agnostic syntax. vRouter specific configuration files must still be created using vRouter syntax. Such scripts are not provided by the characterization tool but should be very easy to write. An example is given in section 3.2.8.1. For the Brocade 5600 vRouter, the interfaces must be configured within the interface section of the `config.boot` and the routes within the protocols section.

So, the user must create `/config/prox/config.x_y.boot` (use case 1, x routes, y next hops) and `/config/prox/config.1_1_2.boot` (use case 2)

The characterization tool copies those different configurations when needed (e.g., when changing the number of next hops between two runs) and reload the SUT. It copies the configurations from `/config/prox/config.x_y.boot` to `/config/config.boot` (for use case 0 and 1) and from `/config/prox/config.1_1_2.boot` to `/config/config.boot`. The script supposes the vRouter to be started when l2tp related message appears in dmesg.

If the vRouter uses a different boot configuration file format, the characterization script must be adapted.

4.4.1 One VM

The prox configuration file must be adapted to the system under test. The proper destination MAC addresses must be inserted in the generated packets. The Brocade 5600 vRouter does (at least by default) not support promiscuous mode. Hence, packets sent with wrong MAC addresses will be silently deleted by the vRouter interfaces.

It's expected that the first four ports on the test generators are connected to the first four ports on the SUT.

4.4.2 Two VMs

The characterization scripts have been written to support one VM. The script supports only simple characterization in the two VM case (e.g., fixed number of routes and next hops). For instance, the characterization script is unable to reload new configurations on two VMs. To run some additional characterizations when the system under test is using two VMs, the vRouters must be manually configured so that they appear from the outside as one VM with eight cores. The `test_description.txt` must be configured so that configuration file is not reloaded (`reload=0`).

```
1; 1; 1; 64; 0; 0
1; 1; 1; 128; 0; 0
1; 1; 1; 256; 0; 0
1; 1; 1; 512; 0; 0
1; 1; 1; 1024; 0; 0
1; 1; 1; 1280; 0; 0
1; 1; 1; 1518; 0; 0
```

It's expected that the first four ports on the test generators are connected to the first four ports on the SUT. Failing to do so will result in bad performance.

5 Running the Characterization

In prox 0.31 with DPDK 2.2.0, the `characterize_vRouter.py` script has an issue related to the inclusion of ierrors in statistics. This requires a change in the script: the line rx+= ierrors should be commented out.

When the vRouter is properly configured, its configuration files copied in `/config/prox` and the `test_description.txt` file created, then the characterization can start. Run

```
./characterize_vRouter.py -r 1
```

The characterization will create up to three results-related files (not all files exist in all use cases):

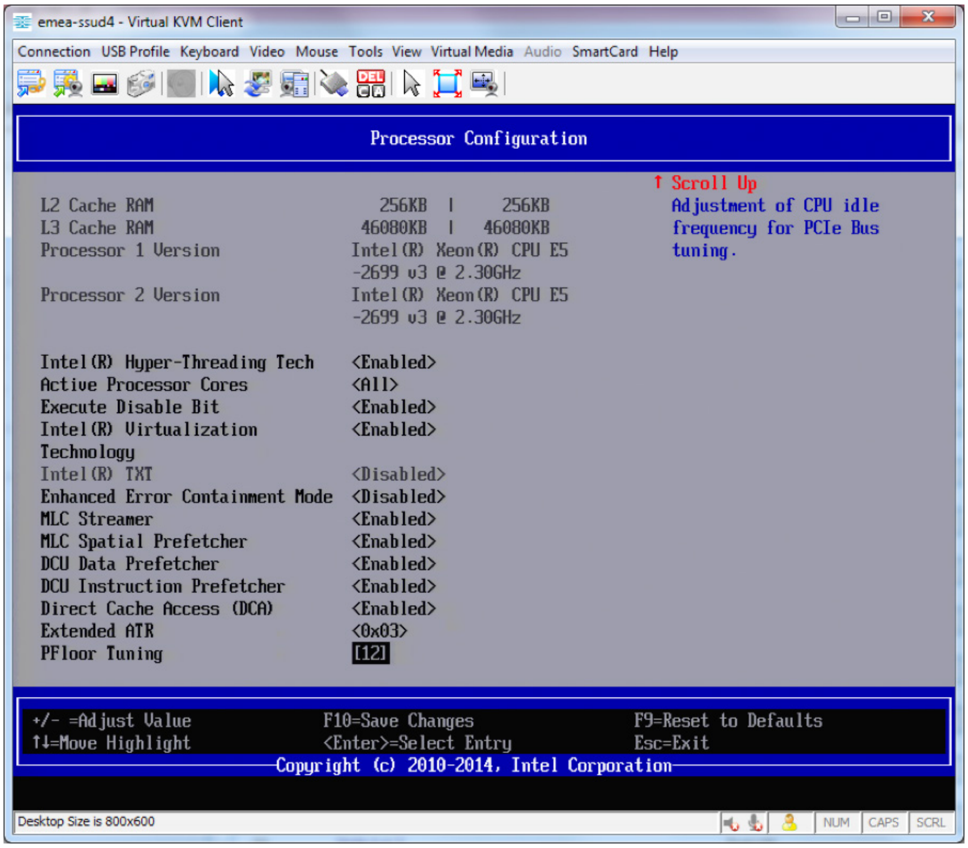
- `minimal_results.txt` contains the results, usually useful to plot use case 1 and 2
- `detailed_results.txt` contains the results to be plotted for use case 2. For use case 0 and 1, it contains all succeeding steps used in the binary search for 0% packet loss; it is used for debugging in use cases 0 and 1
- `all_results.txt` contains all data points for use case 0 and 1. Only useful for debugging (e.g., looking at how many packets were lost and why higher throughput was not obtained).

Those files contain throughput and latency results. They can be plotted using Excel.*

6 BIOS Settings

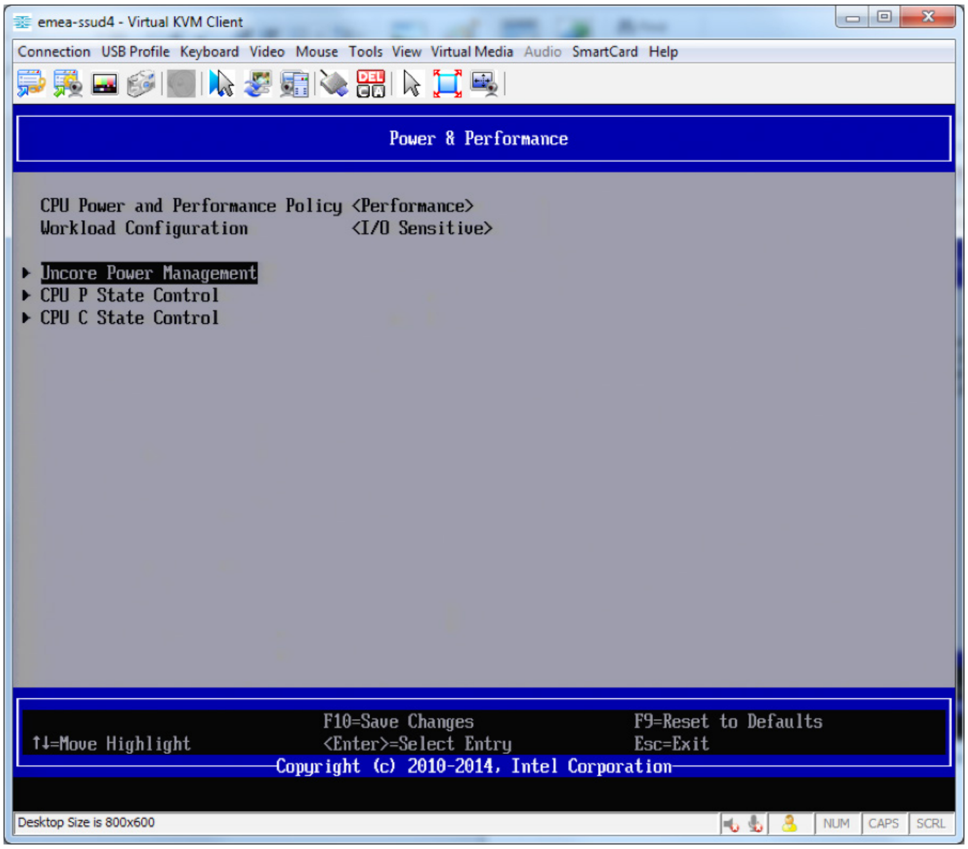
The vRouter system under test configuration requires some specific BIOS settings.

The following screens show the difference compared to default BIOS settings.


emea-ssud4 - Virtual KVM Client
Connection USB Profile Keyboard Video Mouse Tools View Virtual Media Audio SmartCard Help

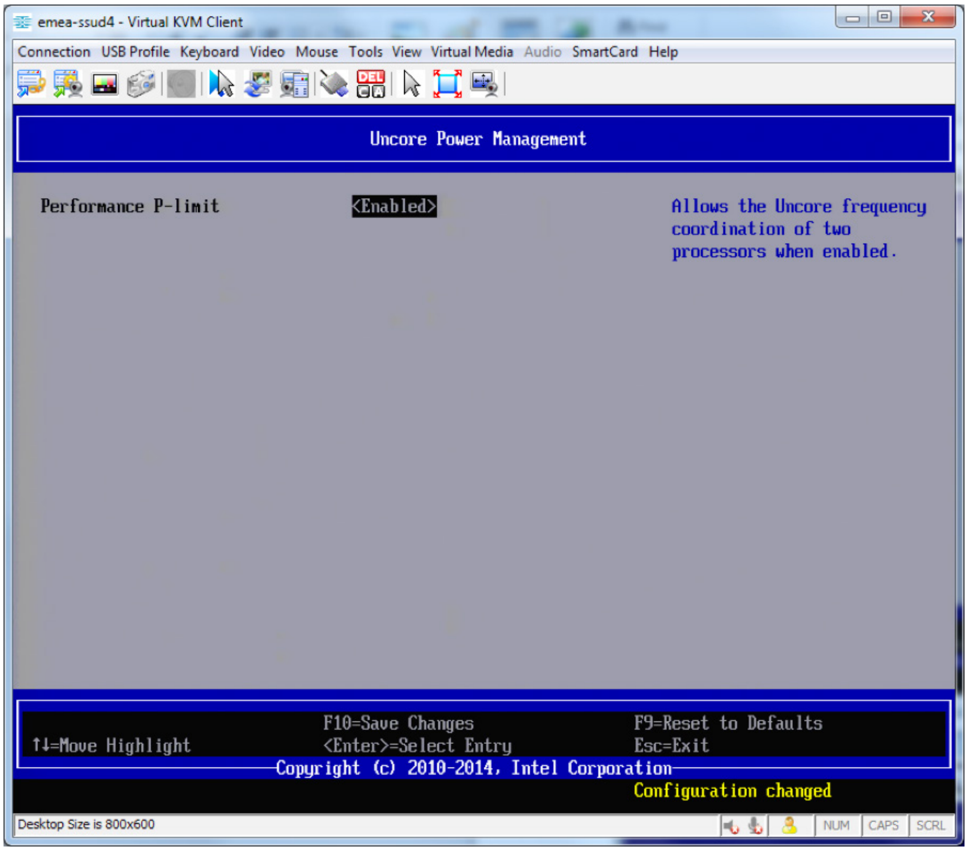
```
                        Processor Configuration

                                                        ↑ Scroll Up
  L2 Cache RAM                       256KB  |     256KB  Adjustment of CPU idle
  L3 Cache RAM                     46080KB  |   46080KB  frequency for PCIe Bus
  Processor 1 Version              Intel(R) Xeon(R) CPU E5  tuning.
                                   -2699 v3 @ 2.30GHz
  Processor 2 Version              Intel(R) Xeon(R) CPU E5
                                   -2699 v3 @ 2.30GHz

  Intel(R) Hyper-Threading Tech    <Enabled>
  Active Processor Cores           <All>
  Execute Disable Bit              <Enabled>
  Intel(R) Virtualization          <Enabled>
  Technology
  Intel(R) TXT                     <Disabled>
  Enhanced Error Containment Mode  <Disabled>
  MLC Streamer                     <Enabled>
  MLC Spatial Prefetcher           <Enabled>
  DCU Data Prefetcher              <Enabled>
  DCU Instruction Prefetcher       <Enabled>
  Direct Cache Access (DCA)        <Enabled>
  Extended ATR                     <0x03>
  PFloor Tuning                    [12]

 +/- =Adjust Value             F10=Save Changes        F9=Reset to Defaults
 ↑↓=Move Highlight             <Enter>=Select Entry    Esc=Exit
                    Copyright (c) 2010-2014, Intel Corporation
```

Desktop Size is 800x600
NUM
CAPS
SCRL


emea-ssud4 - Virtual KVM Client
Connection USB Profile Keyboard Video Mouse Tools View Virtual Media Audio SmartCard Help

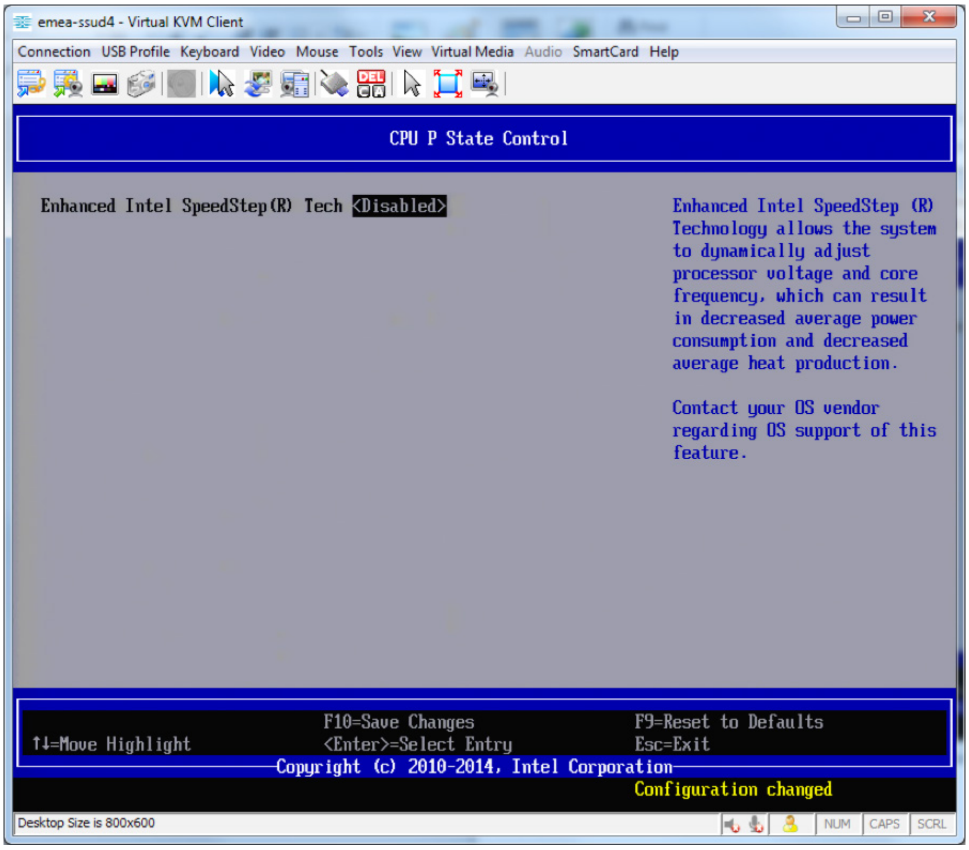
```
                        Power & Performance

  CPU Power and Performance Policy <Performance>
  Workload Configuration           <I/O Sensitive>

▸ Uncore Power Management
▸ CPU P State Control
▸ CPU C State Control

                               F10=Save Changes        F9=Reset to Defaults
 ↑↓=Move Highlight             <Enter>=Select Entry    Esc=Exit
                    Copyright (c) 2010-2014, Intel Corporation
```

Desktop Size is 800x600
NUM
CAPS
SCRL

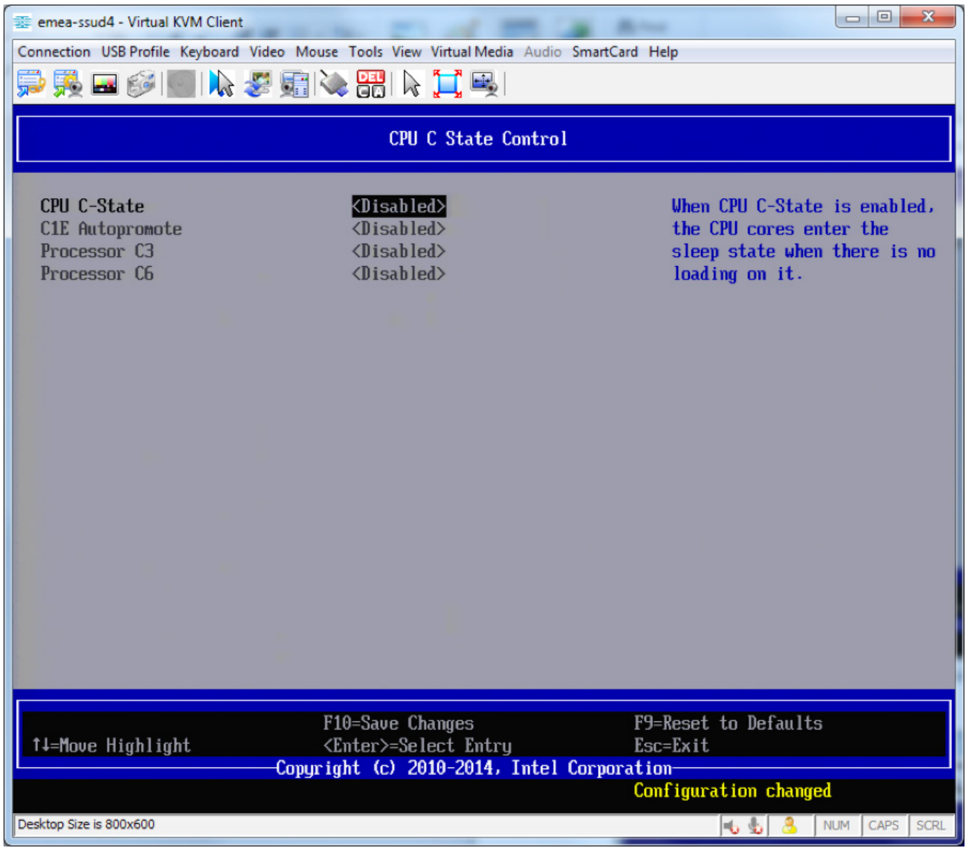

emea-ssud4 - Virtual KVM Client
Connection USB Profile Keyboard Video Mouse Tools View Virtual Media Audio SmartCard Help

```
                         Uncore Power Management

  Performance P-limit            <Enabled>              Allows the Uncore frequency
                                                        coordination of two
                                                        processors when enabled.

                             F10=Save Changes           F9=Reset to Defaults
 ↑↓=Move Highlight           <Enter>=Select Entry       Esc=Exit
                    Copyright (c) 2010-2014, Intel Corporation
                                                        Configuration changed
```

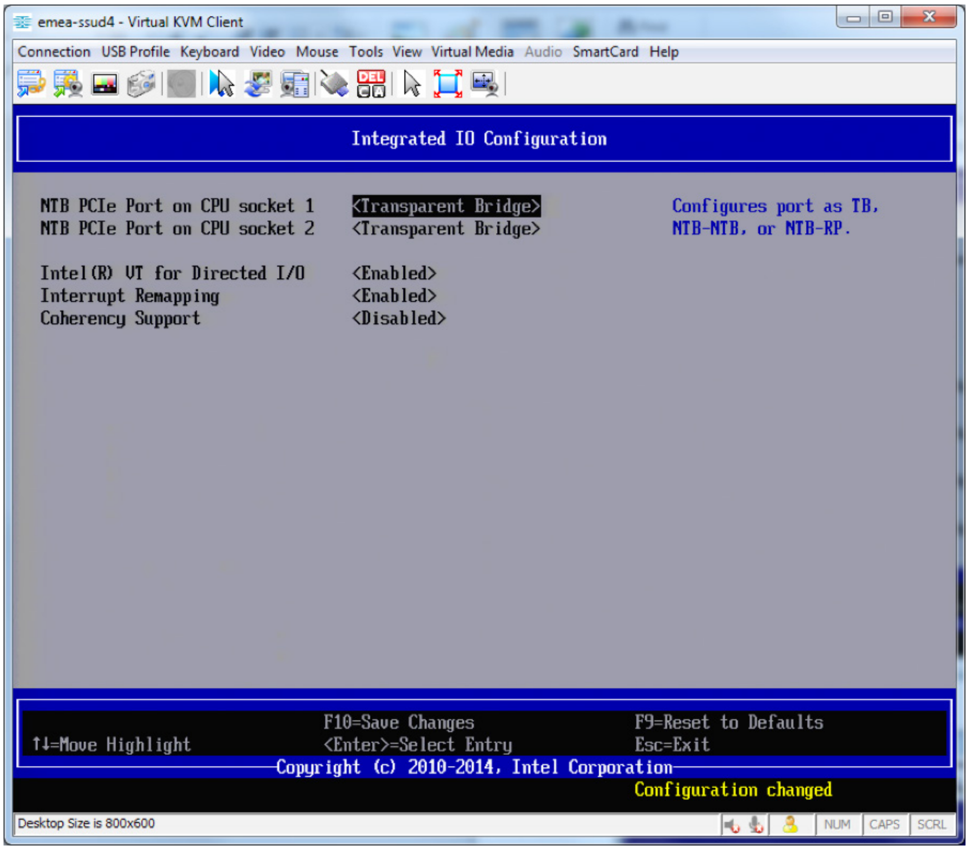
Desktop Size is 800x600
NUM CAPS SCRL


emea-ssud4 - Virtual KVM Client
Connection USB Profile Keyboard Video Mouse Tools View Virtual Media Audio SmartCard Help

```
                            CPU P State Control

  Enhanced Intel SpeedStep(R) Tech <Disabled>           Enhanced Intel SpeedStep (R)
                                                        Technology allows the system
                                                        to dynamically adjust
                                                        processor voltage and core
                                                        frequency, which can result
                                                        in decreased average power
                                                        consumption and decreased
                                                        average heat production.

                                                        Contact your OS vendor
                                                        regarding OS support of this
                                                        feature.

                             F10=Save Changes           F9=Reset to Defaults
 ↑↓=Move Highlight           <Enter>=Select Entry       Esc=Exit
                    Copyright (c) 2010-2014, Intel Corporation
                                                        Configuration changed
```


emea-ssud4 - Virtual KVM Client
Connection USB Profile Keyboard Video Mouse Tools View Virtual Media Audio SmartCard Help
Desktop Size is 800x600
NUM CAPS SCRL

```
                           CPU C State Control

CPU C-State                   <Disabled>              When CPU C-State is enabled,
C1E Autopromote               <Disabled>              the CPU cores enter the
Processor C3                  <Disabled>              sleep state when there is no
Processor C6                  <Disabled>              loading on it.

                        F10=Save Changes              F9=Reset to Defaults
↑↓=Move Highlight       <Enter>=Select Entry          Esc=Exit
                      Copyright (c) 2010-2014, Intel Corporation
                                                      Configuration changed
```


emea-ssud4 - Virtual KVM Client
Connection USB Profile Keyboard Video Mouse Tools View Virtual Media Audio SmartCard Help
Desktop Size is 800x600
NUM CAPS SCRL

```
                        Integrated IO Configuration

NTB PCIe Port on CPU socket 1   <Transparent Bridge>    Configures port as TB,
NTB PCIe Port on CPU socket 2   <Transparent Bridge>    NTB-NTB, or NTB-RP.

Intel(R) VT for Directed I/O    <Enabled>
Interrupt Remapping             <Enabled>
Coherency Support               <Disabled>

                        F10=Save Changes              F9=Reset to Defaults
↑↓=Move Highlight       <Enter>=Select Entry          Esc=Exit
                      Copyright (c) 2010-2014, Intel Corporation
                                                      Configuration changed
```

7 References

PROX https://01.org/intel-data-plane-performance-demonstrators/overview and http://github.com/nvf-crucio/prox

Brocade 5600 vRouter http://www.brocade.com/en/products-services/software-networking/network-functions-virtualization/vrouter.html