VMX tutorial

Ping Song

Table of Content

version	author	date	changelog
Version 0.1	pings@juniper.net	(2015-11-23)	main part of the doc done
Version 0.2	pings@juniper.net	(2015-11-28)	add "references"
Version 0.3	pings@juniper.net	(2016-01-04)	add qemu-kvm params breakdown(TODO)

Kernel-based Virtual Machine (KVM) is a virtualization infrastructure for the Linux kernel that turns it into a hypervisor. It has been gaining industry traction and market share in a wide variety of software deployments in just a few short years. KVM turns to be a very interesting virtualization topic and is well positioned for the future.

Juniper MX86(VMX: Virtual MX) is a virtual version of the MX Series 3D Universal Edge Router that is installed on an industry-standard x86 server running a Linux operating system, applicable third-party software, and the KVM hypervisor. It is one of industry-level implementations of router based on KVM technology.

about This doc :

This doc records some learning and testing notes about KVM/MX86 technology.

- illustration of various installation procedures
- illustration of verification procedure
- illustration of features involved in VMX installation process, mostly from KVM/virtualization technology’s perspecive
- some case studies

This doc can be used as reference when you:

- read the official VMX documents, use this as an "accompanying" doc
- need a quick list of commands to verify a feature or troubleshoot an issue
- when run into issues, need a "working example" as a reference

a "TODO" list:

- breakdown of qemu command parameters
- customerization of the installation script to make it startup friendly
- libvirt API scripting
- DPDK programming
- internal crabs? (flow cache, junos troubleshooting, internal packet flow, etc)

Part 1: VMX installation

Chapter 1. prepare for the installation

1.1. identify current system info

before starting the installation, we need to collect some basic information about the server that VMX is about to be built on. At mininum, make sure the server complies to the "Minimum Hardware and Software Requirements".

pre-installation checklist

- ☐ server manufacturer info
- ☐ operating system
- ☐ cpu and memory
- ☐ NIC/controller
- ☐ NIC driver
- ☐ VT-d/IOMMU
- ☐ kvm/qemu version
- ☐ libvirt version

1.1.1. server Manufacturer/model/SN

this server's manufacturer/model info:

- this server is a `ProLiant BL660c Gen8` server.
- chassis SN is `USE4379WSS`

same info can be acquired from HP `ILO`, but command from ssh session is more convenient.

```
ping@trinity:~$ sudo dmidecode |sed -n '/System Info/,/^$/p'
System Information
        Manufacturer: HP
        Product Name: ProLiant BL660c Gen8          #<------
        Version: Not Specified
        Serial Number: USE4379WSS                   #<------
        UUID: 31393736-3831-5355-4534-333739575353
        Wake-up Type: Power Switch
        SKU Number: 679118-B21
        Family: ProLiant
```

`dmidecode -t 1` will print same info:

```
ping@trinity:~$ sudo dmidecode -t 1
    Handle 0x0100, DMI type 1, 27 bytes
    System Information
            Manufacturer: HP
            Product Name: ProLiant BL660c Gen8
            Version: Not Specified
            Serial Number: USE4379WSS
            UUID: 31393736-3831-5355-4534-333739575353
            Wake-up Type: Power Switch
            SKU Number: 679118-B21
            Family: ProLiant
```

`-t` is handy , but the use of `sed` is a more general way to extract a part of text from long text output of any command.

dmidecode tool

most hardware data can be queried by `dmidecode` and/or `lshw` command. the `dmi type` code of `system info` is 1, so `dmidecode -t 1` will print same info. refer to manpage of `dmidecode` for more info about the usage.

The SMBIOS specification defines the following DMI types:

```
Type   Inf         0   BIOS
   1   System
   2   Baseboard
   3   Chassis
   4   Processor
   5   Memory Controller
   6   Memory Module
   7   Cache
   8   Port Connector
   9   System Slots
  10   On Board Devices
  11   OEM Strings
  12   System Configuration Options
  13   BIOS Language
  14   Group Associations
  16   System Event Log
  16   Physical Memory Array
  17   Memory Device
  18   32-bit Memory Error
  19   Memory Array Mapped Address
  20   Memory Device Mapped Address
  21   Built-in Pointing Device
  22   Portable Battery
  23   System Reset
  24   Hardware Security
  25   System Power Controls
  26   Voltage Probe
  27   Cooling Device
  28   Temperature Probe
  29   Electrical Current Probe
  30   Out-of-band Remote Access
  31   Boot Integrity Services
  32   System Boot
  33   64-bit Memory Error
  34   Management Device
  35   Management Device Component
  36   Management Device Threshold Data
  37   Memory Channel
  38   IPMI Device
  39   Power Supply
  40   Additional Information
  41   Onboard Devices Extended Information
  42   Management Controller Host Interface
```

1.1.2. Operating System

this server's current operation system info:

- linux distribution: ubuntu 14.04.2
- linux kernel in use: `3.19.0-25-generic`

```
ping@trinity:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 14.04.2 LTS  #<------
Release:        14.04
Codename:       trusty

ping@trinity:~$ uname -a
Linux trinity 3.19.0-25-generic #26~14.04.1-Ubuntu SMP Fri Jul 24 21:16:20 UTC 2015
x86_64 x86_64 x86_64 GNU/Linux
              ^^^^^^^^^
```

there are many other alternative commands to print OS version/release info:

```
ping@ubuntu1:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="14.04.1 LTS, Trusty Tahr"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 14.04.1 LTS"
VERSION_ID="14.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"

ping@trinity:~$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04.2 LTS"

ping@trinity:~$ cat /etc/issue
Ubuntu 14.04.2 LTS \n \l

ping@trinity:~$ cat /etc/issue.net
Ubuntu 14.04.2 LTS

ping@ubuntu1:~$ cat /proc/version
Linux version 3.13.0-32-generic (buildd@kissel)
(gcc version 4.8.2 (Ubuntu 4.8.2-19ubuntu1) )
#57-Ubuntu SMP Tue Jul 15 03:51:08 UTC 2014
```

to list all kernels installed currently in the server:

```
ping@trinity:~$ dpkg --get-selections | grep linux-image
linux-image-3.13.0-32-generic                   install     #<------
linux-image-3.16.0-30-generic                   install     #<------
linux-image-3.19.0-25-generic                   install     #<------
linux-image-extra-3.13.0-32-generic             install
linux-image-extra-3.16.0-30-generic             install
linux-image-extra-3.19.0-25-generic             install
linux-image-generic-lts-utopic                  install
```

In this server multiple verions of linux kernel have been installed, including our target kernel `3.13.0-32-generic`. There is no need to install the the kernel again - we only need to select the right one and reboot the system with it.

1.1.3. cpu and memory

this server's CPU and memory info:

- CPU model: Intel® Xeon® CPU E5-4627 v2 @ 3.30GHz (Ivy bridge)
- number of "CPU processors": 4
- number of "physical cores" per CPU processor: 8
- VT-x (not VT-d) enabled
- This CPU does not support HT(HyperThreading), see this spec
- 32 32G DDR3 memory, 500+G total memory available

```
ping@trinity:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                32           #<---- ①
On-line CPU(s) list:   0-31
Thread(s) per core:    1            #<---- ②
Core(s) per socket:    8
Socket(s):             4            #<---- ④
NUMA node(s):          4
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 62
Stepping:              4
CPU MHz:               3299.797
BogoMIPS:              6605.01
Virtualization:        VT-x         #<---- ③
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              16384K
NUMA node0 CPU(s):     0-7
NUMA node1 CPU(s):     8-15
NUMA node2 CPU(s):     16-23
NUMA node3 CPU(s):     24-31
```

① total number of CPU cores

② hyperthreading is not enabled or not supported in this server

③ VT-x is enabled

④ max number of sockets available (to hold CPU) in the server

```
ping@trinity:~$ grep -m1 "model name"  /proc/cpuinfo
model name      : Intel(R) Xeon(R) CPU E5-4627 v2 @ 3.30GHz   #<------

ping@trinity:~$ cat /proc/cpuinfo | sed -e '/^$/,$d'
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 62
model name      : Intel(R) Xeon(R) CPU E5-4627 v2 @ 3.30GHz   #<------
stepping        : 4
microcode       : 0x427
cpu MHz         : 3299.797
cache size      : 16384 KB
physical id     : 0
siblings        : 8
core id         : 0
cpu cores       : 8
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr  ①
                  pge mca cmov pat pse36 clflush dts acpi mmx fxsr
                  sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp   ②
                  lm constant_tsc arch_perfmon pebs bts rep_good
                  nopl xtopology nonstop_tsc aperfmperf eagerfpu
                  pni pclmulqdq dtes64 monitor ds_cpl vmx smx est   ③
                  tm2 ssse3 cx16 xtpr pdcm pcid dca sse4_1 sse4_2
                  x2apic popcnt tsc_deadline_timer aes xsave avx
                  f16c rdrand lahf_lm ida arat epb pln pts dtherm   ④
                  tpr_shadow vnmi flexpriority ept vpid fsgsbase
                  smep erms xsaveopt
bugs            :
bogomips        : 6599.59
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

① `pse` flag indicate 2M hugepage support

② `pdpe1gb` flag indicate 1G hugepage support

③ `vmx` flag indicate VT-x capability

④ `rdrand` flag, required by current VMX implementation, supported in Ivy Bridge CPU and not in Sandy Bridge CPU

memory

```
ping@trinity: free -h
             total       used       free     shared    buffers     cached
Mem:          503G        13G       490G       1.5M       176M       6.9G
-/+ buffers/cache:       5.9G       497G
Swap:          14G         0B        14G
```

`dmidecode` *version of the cpu information captured is shown below:*

```
ping@trinity: sudo dmidecode -t 4
# dmidecode 2.12
SMBIOS 2.8 present.

    Handle 0x0400, DMI type 4, 42 bytes
    Processor Information
            Socket Designation: Proc 1
            Type: Central Processor
            Family: Xeon
            Manufacturer: Intel
            ID: E4 06 03 00 FF FB EB BF
            Signature: Type 0, Family 6, Model 62, Stepping 4
            Flags:
                    FPU (Floating-point unit on-chip)
                    VME (Virtual mode extension)
                    DE (Debugging extension)
                    PSE (Page size extension)
                    TSC (Time stamp counter)
                    MSR (Model specific registers)
                    PAE (Physical address extension)
                    MCE (Machine check exception)
                    CX8 (CMPXCHG8 instruction supported)
                    APIC (On-chip APIC hardware supported)
                    SEP (Fast system call)
                    MTRR (Memory type range registers)
                    PGE (Page global enable)
                    MCA (Machine check architecture)
                    CMOV (Conditional move instruction supported)
                    PAT (Page attribute table)
                    PSE-36 (36-bit page size extension)
                    CLFSH (CLFLUSH instruction supported)
                    DS (Debug store)
                    ACPI (ACPI supported)
                    MMX (MMX technology supported)
                    FXSR (FXSAVE and FXSTOR instructions supported)
                    SSE (Streaming SIMD extensions)
                    SSE2 (Streaming SIMD extensions 2)
                    SS (Self-snoop)
                    HTT (Multi-threading)
                    TM (Thermal monitor supported)
```

```
                        PBE (Pending break enabled)
                Version:  Intel(R) Xeon(R) CPU E5-4627 v2 @ 3.30GHz
                Voltage: 1.4 V
                External Clock: 100 MHz
                Max Speed: 4800 MHz
                Current Speed: 3300 MHz
                Status: Populated, Enabled
                Upgrade: Socket LGA2011
                L1 Cache Handle: 0x0710
                L2 Cache Handle: 0x0720
                L3 Cache Handle: 0x0730
                Serial Number: Not Specified
                Asset Tag: Not Specified
                Part Number: Not Specified
                Core Count: 8
                Core Enabled: 8
                Thread Count: 8
                Characteristics:
                        64-bit capable

        Handle 0x0401, DMI type 4, 42 bytes
        Processor Information
                Socket Designation: Proc 2
                Type: Central Processor
                ......

        Handle 0x0402, DMI type 4, 42 bytes
        Processor Information
                Socket Designation: Proc 3
                ......

        Handle 0x0403, DMI type 4, 42 bytes
        Processor Information
                Socket Designation: Proc 4
                ......
```

`dmidecode` *version of the memory data is:* `dmidecode -t 17`*:*

```
ping@trinity: sudo dmidecode -t 17
# dmidecode 2.12
SMBIOS 2.8 present.

    Handle 0x1100, DMI type 17, 40 bytes
    Memory Device
            Array Handle: 0x1000
            Error Information Handle: Not Provided
            Total Width: 72 bits
            Data Width: 64 bits
            Size: 32 GB         #<------
            Form Factor: DIMM
            Set: None
```

```
        Locator: PROC  1 DIMM  1
        Bank Locator: Not Specified
        Type: DDR3
        Type Detail: Synchronous
        Speed: 1866 MHz
        Manufacturer: HP
        Serial Number: Not Specified
        Asset Tag: Not Specified
        Part Number: 712384-081
        Rank: 4
        Configured Clock Speed: 1866 MHz
        Minimum voltage:  1.500 V
        Maximum voltage:  1.500 V
        Configured voltage:  1.500 V

Handle 0x1101, DMI type 17, 40 bytes
Memory Device
        Array Handle: 0x1000
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: No Module Installed
        Form Factor: DIMM
        Set: 1
        Locator: PROC  1 DIMM  2
        Bank Locator: Not Specified
        Type: DDR3
        Type Detail: Synchronous
        Speed: Unknown
        Manufacturer: UNKNOWN
        Serial Number: Not Specified
        Asset Tag: Not Specified
        Part Number: NOT AVAILABLE
        Rank: Unknown
        Configured Clock Speed: Unknown
        Minimum voltage:  Unknown
        Maximum voltage:  Unknown
        Configured voltage:  Unknown

Handle 0x1102, DMI type 17, 40 bytes
Memory Device
        ......
        Set: 2
        Locator: PROC  1 DIMM  3
        ......

......

Handle 0x111F, DMI type 17, 40 bytes
Memory Device
        Array Handle: 0x1003
```

```
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: 32 GB
        Form Factor: DIMM
        Set: 31
        Locator: PROC  4 DIMM  8
        Bank Locator: Not Specified
        Type: DDR3
        Type Detail: Synchronous
        Speed: 1866 MHz
        Manufacturer: HP
        Serial Number: Not Specified
        Asset Tag: Not Specified
        Part Number: 712384-081
        Rank: 4
        Configured Clock Speed: 1866 MHz
        Minimum voltage:  1.500 V
        Maximum voltage:  1.500 V
        Configured voltage:  1.500 V
```

1.1.4. network adapter/controller

This server's adapter/NIC/controller info:

- two type of HP NIC adapters were equiped:
  - HP 560FLB adapter,
  - HP 560M adapter,
- Intel 82599 10GE controller
- ixgbe kernel driver module that came with linux kernel is in use
  - SR-IOV supported
  - by default VF has not yet been configured
- adapter/interface/PCI address mapping table of all physical network interfaces in this server:

Table 1. server interface table

NO.	"PCI address"	adapter	interface name
1	02:00.0	560FLB	em9
2	02:00.1	560FLB	em10
3	06:00.0	560M	p2p1
4	06:00.1	560M	p2p2
5	21:00.0	560FLB	em1

NO.	"PCI address"	adapter	interface name
6	21:00.1	560FLB	em2
7	23:00.0	560M	p3p1
8	23:00.1	560M	p3p2

[lspci] `lspci` is a linux utility for displaying information about PCI buses in the system and devices connected to them.

```
ping@trinity:~$ lspci | grep -i ethernet
02:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
02:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
06:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
06:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
21:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
21:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
23:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
23:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
```

the prefix number string of each line is a "PCI address":

```
02:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
 ^  ^ ^
 |  | |
 |  | function (port): 0-7
 |  |
 | slot (NIC): 0-1f
 |
bus: 0-ff
```

there is also a "domain" before "bus:slot.function", usually with a value `0000` and is ignored. see `man lspci` option `-s` and `-D`.

to get more details of each device (NIC port) via PCI address:

```
ping@trinity:~$ sudo lspci -vs 02:00.0
02:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
```

```
Connection (rev 01) ②
    Subsystem: Hewlett-Packard Company Ethernet 10Gb 2-port 560FLB Adapter ①
    Flags: bus master, fast devsel, latency 0, IRQ 64
    Memory at ef700000 (32-bit, non-prefetchable) [size=1M]
    I/O ports at 4000 [size=32]
    Memory at ef6f0000 (32-bit, non-prefetchable) [size=16K]
    [virtual] Expansion ROM at ef400000 [disabled] [size=512K]
    Capabilities: [40] Power Management version 3
    Capabilities: [50] MSI: Enable- Count=1/1 Maskable+ 64bit+
    Capabilities: [70] MSI-X: Enable+ Count=64 Masked-
    Capabilities: [a0] Express Endpoint, MSI 00
    Capabilities: [e0] Vital Product Data
    Capabilities: [100] Advanced Error Reporting
    Capabilities: [140] Device Serial Number 00-00-00-ff-ff-00-00-00
    Capabilities: [150] Alternative Routing-ID Interpretation (ARI)
    Capabilities: [160] Single Root I/O Virtualization (SR-IOV)            ④
    Kernel driver in use: ixgbe                                            ③

ping@trinity:~$ sudo lspci -vs 06:00.0
06:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01) ②
    Subsystem: Hewlett-Packard Company Ethernet 10Gb 2-port 560M Adapter ①
    Physical Slot: 2
    Flags: bus master, fast devsel, latency 0, IRQ 136
    Memory at eff00000 (32-bit, non-prefetchable) [size=1M]
    I/O ports at 6000 [size=32]
    Memory at efef0000 (32-bit, non-prefetchable) [size=16K]
    [virtual] Expansion ROM at efc00000 [disabled] [size=512K]
    Capabilities: [40] Power Management version 3
    Capabilities: [50] MSI: Enable- Count=1/1 Maskable+ 64bit+
    Capabilities: [70] MSI-X: Enable+ Count=64 Masked-
    Capabilities: [a0] Express Endpoint, MSI 00
    Capabilities: [e0] Vital Product Data
    Capabilities: [100] Advanced Error Reporting
    Capabilities: [140] Device Serial Number 00-00-00-ff-ff-00-00-00
    Capabilities: [150] Alternative Routing-ID Interpretation (ARI)
    Capabilities: [160] Single Root I/O Virtualization (SR-IOV)          ④
    Kernel driver in use: ixgbe                                          ③

ping@trinity:~$ sudo lspci -vs 21:* | grep -iE "controller|adapter"
21:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01) ②
        Subsystem: Hewlett-Packard Company Ethernet 10Gb 2-port 560FLB Adapter ①
21:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
        Subsystem: Hewlett-Packard Company Ethernet 10Gb 2-port 560FLB Adapter

ping@trinity:~$ sudo lspci -vs 23:* | grep -iE "controller|adapter"
23:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
        Subsystem: Hewlett-Packard Company Ethernet 10Gb 2-port 560M Adapter
```

```
23:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
        Subsystem: Hewlett-Packard Company Ethernet 10Gb 2-port 560M Adapter
```

① adapter vendor info

② controller vendor info

③ current driver in use is ixgbe

④ driver kernel module support SR-IOV feature

1.1.5. ixgbe kernel driver

- current ixgbe driver of this server is the one that came with linux kernel `3.19.1`:
  - ixgbe version 4.0.1-k
  - support following capabilities:
    - `max_vfs` parameter: max 63 VF allowed per port (default 0: VF won't be enabled by default)
    - `allow_unsupported_sfp` parameter: unsupported/untested SFP allowed
    - `debug` option
- the README.txt file from VMX installation package contains more info about the ixgbe driver

```
ping@trinity:~$ cat /sys/module/ixgbe/version
4.0.1-k
```

```
ping@trinity:~$ modinfo ixgbe
filename:
  /lib/modules/3.19.0-25-generic/kernel/drivers/net/ethernet/intel/ixgbe/ixgbe.ko
version:        4.0.1-k     #<------
license:        GPL
description:    Intel(R) 10 Gigabit PCI Express Network Driver
author:         Intel Corporation, <linux.nics@intel.com>
srcversion:     44CBFE422F8BAD726E61653
alias:          pci:v00008086d000015ABsv*sd*bc*sc*i*
alias:          pci:v00008086d000015AAsv*sd*bc*sc*i*
alias:          pci:v00008086d00001563sv*sd*bc*sc*i*
alias:          pci:v00008086d00001560sv*sd*bc*sc*i*
alias:          pci:v00008086d0000154Asv*sd*bc*sc*i*
alias:          pci:v00008086d00001557sv*sd*bc*sc*i*
alias:          pci:v00008086d00001558sv*sd*bc*sc*i*
alias:          pci:v00008086d0000154Fsv*sd*bc*sc*i*
alias:          pci:v00008086d0000154Dsv*sd*bc*sc*i*
alias:          pci:v00008086d00001528sv*sd*bc*sc*i*
alias:          pci:v00008086d000010F8sv*sd*bc*sc*i*
alias:          pci:v00008086d0000151Csv*sd*bc*sc*i*
alias:          pci:v00008086d00001529sv*sd*bc*sc*i*
```

```
alias:          pci:v00008086d0000152Asv*sd*bc*sc*i*
alias:          pci:v00008086d000010F9sv*sd*bc*sc*i*
alias:          pci:v00008086d00001514sv*sd*bc*sc*i*
alias:          pci:v00008086d00001507sv*sd*bc*sc*i*
alias:          pci:v00008086d000010FBsv*sd*bc*sc*i*
alias:          pci:v00008086d00001517sv*sd*bc*sc*i*
alias:          pci:v00008086d000010FCsv*sd*bc*sc*i*
alias:          pci:v00008086d000010F7sv*sd*bc*sc*i*
alias:          pci:v00008086d00001508sv*sd*bc*sc*i*
alias:          pci:v00008086d000010DBsv*sd*bc*sc*i*
alias:          pci:v00008086d000010F4sv*sd*bc*sc*i*
alias:          pci:v00008086d000010E1sv*sd*bc*sc*i*
alias:          pci:v00008086d000010F1sv*sd*bc*sc*i*
alias:          pci:v00008086d000010ECsv*sd*bc*sc*i*
alias:          pci:v00008086d000010DDsv*sd*bc*sc*i*
alias:          pci:v00008086d0000150Bsv*sd*bc*sc*i*
alias:          pci:v00008086d000010C8sv*sd*bc*sc*i*
alias:          pci:v00008086d000010C7sv*sd*bc*sc*i*
alias:          pci:v00008086d000010C6sv*sd*bc*sc*i*
alias:          pci:v00008086d000010B6sv*sd*bc*sc*i*
depends:        mdio,ptp,dca,vxlan
intree:         Y
vermagic:       3.19.0-25-generic SMP mod_unload modversions
signer:         Magrathea: Glacier signing key
sig_key:        6A:AA:11:D1:8C:2D:3A:40:B1:B4:DB:E5:BF:8A:D6:56:DD:F5:18:38
sig_hashalgo:   sha512
parm:           max_vfs:Maximum number of virtual functions to allocate per
    physical function - default is zero and maximum value is 63. (Deprecated)
    (uint)
parm:           allow_unsupported_sfp:Allow unsupported and untested SFP+
    modules on 82599-based adapters (uint)
parm:           debug:Debug level (0=none,...,16=all) (int)
```

reference

- Intel® 82599 SR-IOV Driver Companion Guide

1.1.6. VT-d/IOMMU

Intel `VT-d` feaure is supported in this server's current OS kernel.

```
ping@trinity:~$ less /boot/config-3.19.0-25-generic | grep -i iommu
CONFIG_GART_IOMMU=y
CONFIG_CALGARY_IOMMU=y
CONFIG_CALGARY_IOMMU_ENABLED_BY_DEFAULT=y
CONFIG_IOMMU_HELPER=y
CONFIG_VFIO_IOMMU_TYPE1=m
CONFIG_IOMMU_API=y
CONFIG_IOMMU_SUPPORT=y          #<------
CONFIG_AMD_IOMMU=y
CONFIG_AMD_IOMMU_STATS=y
CONFIG_AMD_IOMMU_V2=m
CONFIG_INTEL_IOMMU=y            #<------
# CONFIG_INTEL_IOMMU_DEFAULT_ON is not set
CONFIG_INTEL_IOMMU_FLOPPY_WA=y
# CONFIG_IOMMU_DEBUG is not set
# CONFIG_IOMMU_STRESS is not set

ping@trinity:~$ grep -i remap /boot/config-3.13.0-32-generic
CONFIG_HAVE_IOREMAP_PROT=y
CONFIG_IRQ_REMAP=y              #<------

ping@ubuntu1:~$ grep -i pci_stub /boot/config-3.13.0-32-generic
CONFIG_PCI_STUB=m               #<------
```

a more "general" way to print kernel info is: `less /boot/config-`uname -r``

1.1.7. kvm/qemu software version

- qemu version 2.0.0 (Debian 2.0.0+dfsg-2ubuntu1.19)
- kvm version (same as linux kernel version)

This looks OK comparing with the "Minimum Hardware and Software" from the release note:

> VirtualizationQEMU-KVM 2.0.0+dfsg-2ubuntu1.11 or later

To verify qemu/kvm version and hardware acceleration support:

1. qemu version:

   ```
   ping@trinity:~$ qemu-system-x86_64 --version
   QEMU emulator version 2.0.0 (Debian 2.0.0+dfsg-2ubuntu1.19), Copyright (c) 2003-
   2008 Fabrice Bellard
   ```

   or

```
ping@trinity:~$ kvm --version
QEMU emulator version 2.0.0 (Debian 2.0.0+dfsg-2ubuntu1.19), Copyright (c) 2003-
2008 Fabrice Bellard
```

2. KVM kernel module info:

```
ping@trinity:~$ modinfo kvm
filename:       /lib/modules/3.19.0-25-generic/kernel/arch/x86/kvm/kvm.ko
license:        GPL
author:         Qumranet
srcversion:     F58A0F8858A02EFA0549DE5
depends:
intree:         Y
vermagic:       3.19.0-25-generic SMP mod_unload modversions
signer:         Magrathea: Glacier signing key
sig_key:        6A:AA:11:D1:8C:2D:3A:40:B1:B4:DB:E5:BF:8A:D6:56:DD:F5:18:38
sig_hashalgo:   sha512
parm:           allow_unsafe_assigned_interrupts:Enable device assignment on
platforms without interrupt remapping support. (bool)
parm:           ignore_msrs:bool
parm:           min_timer_period_us:uint
parm:           tsc_tolerance_ppm:uint
```

3. HW acceleration OK:

```
ping@trinity:~$ kvm-ok
INFO: /dev/kvm exists
KVM acceleration can be used

ping@MX86-host-BL660C-B1: ls -l /dev/kvm
crw-rw---- 1 root kvm 10, 232 Nov  5 11:35 /dev/kvm
```

The development of qemu software is very active, and code changes (for bug fix, security update, new features) and new releases appear frequently. For ubuntu linux the changelogs are available here

1.1.8. libvirt

- current libvirt version is 1.2.2
- this needs to be upgraded to required libvirt version.

```
ping@ubuntu:~$ libvirtd --version
libvirtd (libvirt) 1.2.2
```

libvirt 1.2.2 misses some of bug fixes of features that are important for VMX. One of these are `numatune`, which is used to "pin" vCPUs of guest VM to physcial CPUs all in one NUMA node, hence reducing the `NUMA misses` that is one of the main contributor of performance impact.

Now after identifying the server HW/SW configuration, the checklist looks:

pre-installation checklist

- ☐ server manufacturer info
- ☑ operating system
- ☐ cpu and memory
- ☐ NIC/controller
- ☑ NIC driver
- ☑ VT-d/IOMMU
- ☐ kvm/qemu version
- ☑ libvirt version

those checked mark indicate those items that do not meet the requirement and need some change.

1.2. adjust the system

after collecting the server's current configuration, we need to change some setting according to the Preparing the System to Install vMX portion in the VMX document.

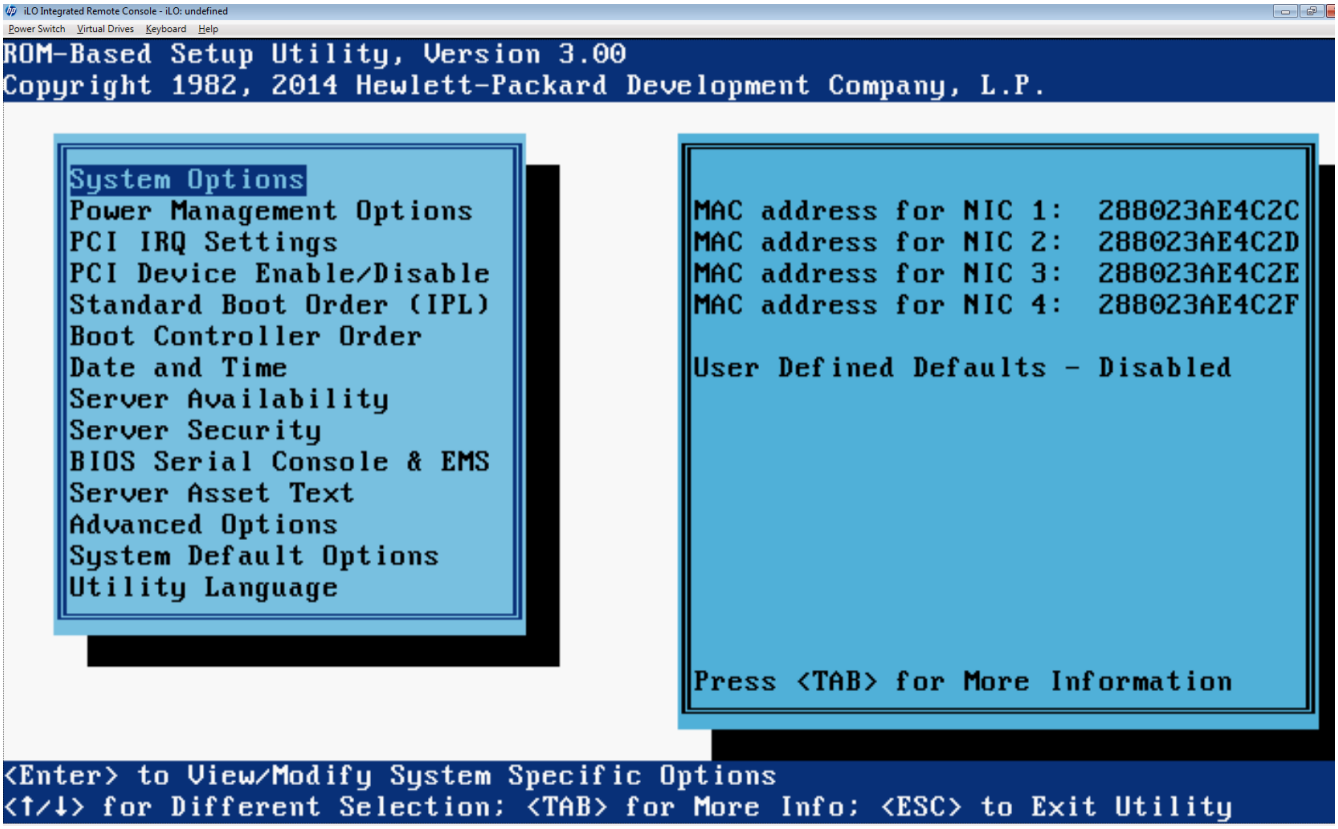
1.2.1. BIOS setting

following (Intel) virtualization features are required to setup VMX and need to be enabled from within BIOS, if not yet.

- VT-x
- VT-d
- SR-IOV
- HyperThreading

the requirment may change in different VMX release

enter BIOS setting:


iLO Integrated Remote Console - iLO: undefined
Power Switch Virtual Drives Keyboard Help
ROM-Based Setup Utility, Version 3.00
Copyright 1982, 2014 Hewlett-Packard Development Company, L.P.
System Options
Power Management Options
PCI IRQ Settings
PCI Device Enable/Disable
Standard Boot Order (IPL)
Boot Controller Order
Date and Time
Server Availability
Server Security
BIOS Serial Console & EMS
Server Asset Text
Advanced Options
System Default Options
Utility Language
MAC address for NIC 1: 288023AE4C2C
MAC address for NIC 2: 288023AE4C2D
MAC address for NIC 3: 288023AE4C2E
MAC address for NIC 4: 288023AE4C2F
User Defined Defaults - Disabled
Press <TAB> for More Information
<Enter> to View/Modify System Specific Options
<↑/↓> for Different Selection; <TAB> for More Info; <ESC> to Exit Utility

even in same hardware, different version of BIOS might look a little bit different. In this system we have BIOS version "I32"

Here are more details of current BIOS info in this server:

```
labroot@MX86-host-BL660C-B1:~/vmx_20141216$ sudo dmidecode
# dmidecode 2.12
SMBIOS 2.8 present.
227 structures occupying 7860 bytes.
Table at 0xBFBDB000.
```

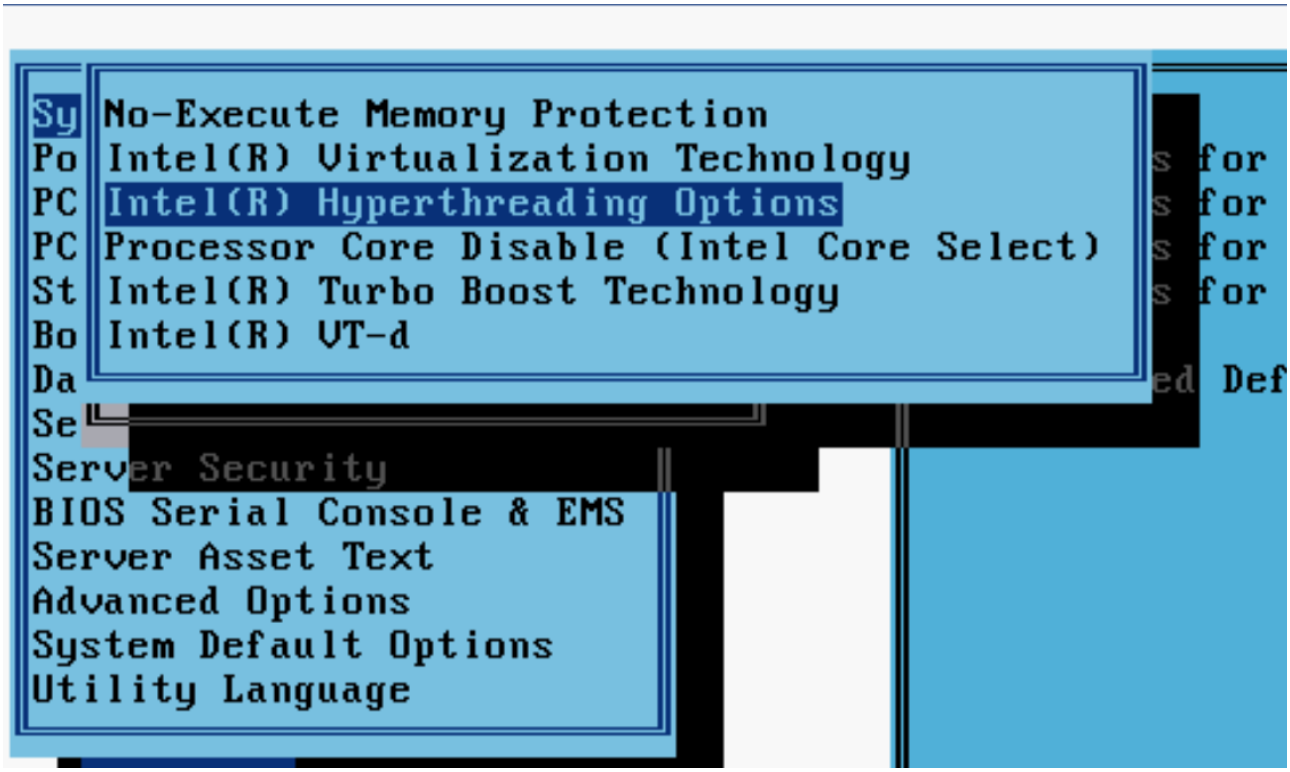
```
Handle 0x0000, DMI type 0, 24 bytes
BIOS Information
        Vendor: HP
        Version: I32        #<------
        Release Date: 02/10/2014
        Address: 0xF0000
        Runtime Size: 64 kB
        ROM Size: 8192 kB
        Characteristics:
                PCI is supported
                PNP is supported
                BIOS is upgradeable
                BIOS shadowing is allowed
                ESCD support is available
                Boot from CD is supported
                Selectable boot is supported
                EDD is supported
                5.25"/360 kB floppy services are supported (int 13h)
                5.25"/1.2 MB floppy services are supported (int 13h)
                3.5"/720 kB floppy services are supported (int 13h)
                Print screen service is supported (int 5h)
                8042 keyboard services are supported (int 9h)
                Serial services are supported (int 14h)
                Printer services are supported (int 17h)
                CGA/mono video services are supported (int 10h)
                ACPI is supported
                USB legacy is supported
                BIOS boot specification is supported
                Function key-initiated network boot is supported
                Targeted content distribution is supported
        Firmware Revision: 1.51
```

1. VT-x/VT-d/HyperThreading

   BIOS | System Options | Intel® Virtualization technology

   BIOS | System Options | Intel® VT-d

BIOS | System Options | Intel® HyperThreading Options



The new VMX releases requires HyperThreading to be enabled to support flow cache feature. The installation script will abort if HT is not enabled, see troubleshooting installation script for more detail of this issue. To install VMX, Either modifying the script to disable HT1 calculation/verification, or installing VMX manually.

According to "Minimum Hardware and Software Requirements":

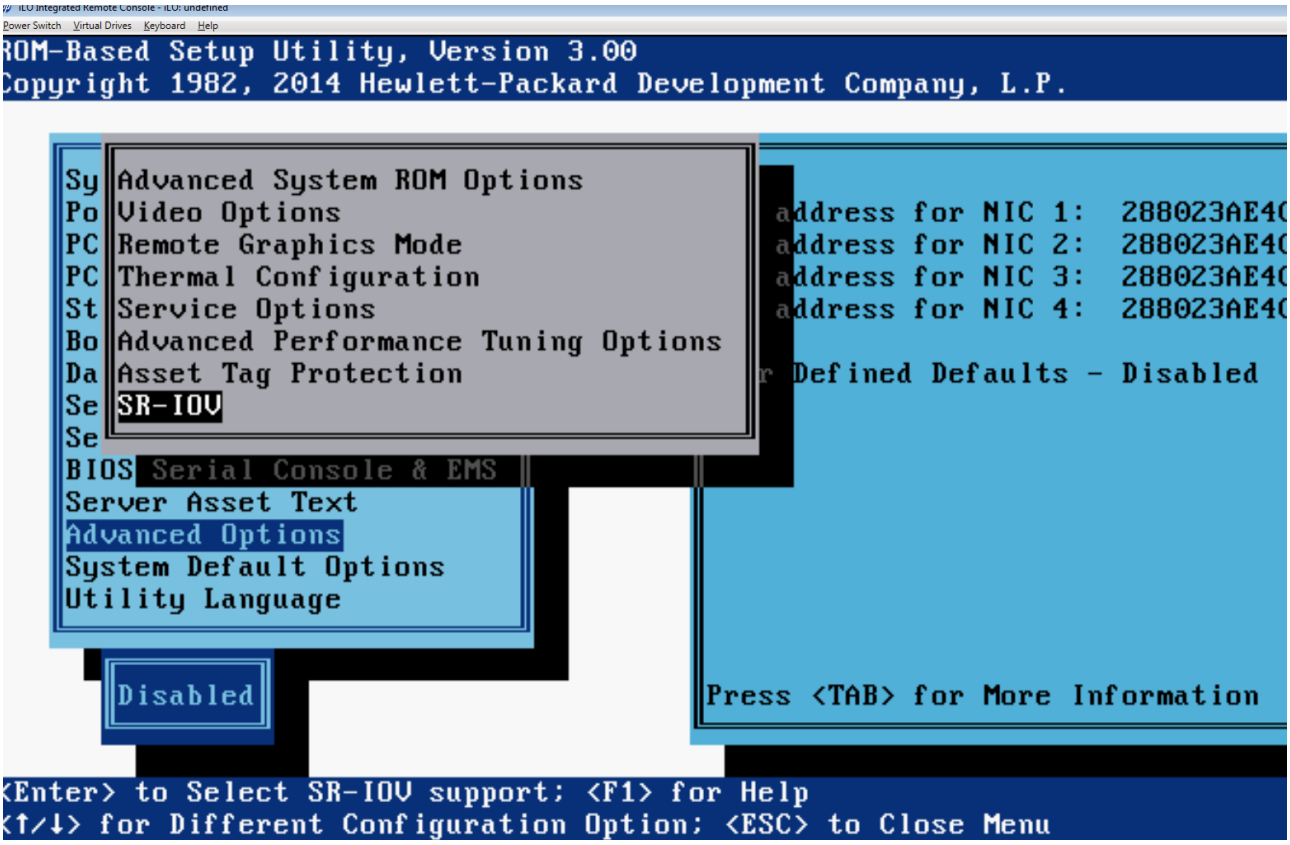
> - For lab simulation and low performance (less than 100 Mbps) use cases, any x86 processor (Intel or AMD) with VT-d capability.
> - For all other use cases, Intel Ivy Bridge processors or later are required. Example of Ivy Bridge processor: Intel Xeon E5-2667 v2 @ 3.30 GHz 25 MB Cache
> - For single root I/O virtualization (SR-IOV) NIC type, use Intel 82599-based PCI-Express cards (10 Gbps) and Ivy Bridge processors.

These statements indicate some current implementation info:

- lab simulation/low performance ⇒ virtio
- all other use cases ⇒ high performance ⇒ SRIOV
- "Ivy Bridge CPU" is required for VMX running SR-IOV

2. SR-IOV

   BIOS | Advanced Options | SR-IOV



1.2.2. enable iommu/VT-d

to enable VT-d we need to change kernel boot parameters. Here is the steps:

1. Make sure the `/etc/default/grub` file contains this line:

   ```
   GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=on"
   ```

2. If not, add it:

   ```
   echo 'GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=on"' >> /etc/default/grub
   ```

   So it looks like:

   ```
   ping@trinity:~$ grep -i iommu /etc/default/grub
   GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=on"
   ```

3. Run `sudo update-grub` to update grub
4. reboot system to make it start with configured kernel parameters

```
sudo reboot
```

when using 'echo', be careful to use >> instead of >. > will "overwrite" the whole file with whatever echoed, instead of "append". in case that happens, correct it with instruction here.

instead of checking the grub config file, looking at parameters passed to the kernel at the time it is started might be more accurate:

```
ping@matrix:~$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-3.19.0-25-generic.efi.signed
root=UUID=875bb72c-c5de-4329-af48-55af85f26398 ro intel_iommu=on
pci=realloc
```

in this example, we are sure the current kernel has IOMMU enabled.

In a system that didn't enable IOMMU, after enabling it and restart, you will notice kernel logs similar to below captures:

```
ping@Compute24:~$ dmesg -T | grep -iE "iommu|dmar"
[Thu Dec 17 09:07:22 2015] Command line: BOOT_IMAGE=/vmlinuz-3.16.0-30-generic
    root=/dev/mapper/Compute24--vg-root ro intel_iommu=on
    crashkernel=384M-2G:64M,2G-16G:128M,16G-:256M
[Thu Dec 17 09:07:22 2015] ACPI: DMAR 0x00000000BDDAB840 000718 (v01 HP
    ProLiant 00000001 \xffffffd2?   0000162E)
[Thu Dec 17 09:07:22 2015] Kernel command line:
    BOOT_IMAGE=/vmlinuz-3.16.0-30-generic root=/dev/mapper/Compute24--vg-root ro
    intel_iommu=on crashkernel=384M-2G:64M,2G-16G:128M,16G-:256M
[Thu Dec 17 09:07:22 2015] Intel-IOMMU: enabled
[Thu Dec 17 09:07:22 2015] dmar: Host address width 46
[Thu Dec 17 09:07:22 2015] dmar: DRHD base: 0x000000f34fe000 flags: 0x0
[Thu Dec 17 09:07:22 2015] dmar: IOMMU 0: reg_base_addr f34fe000 ver 1:0 cap
d2078c106f0466 ecap f020de
[Thu Dec 17 09:07:22 2015] dmar: DRHD base: 0x000000f7efe000 flags: 0x0
[Thu Dec 17 09:07:22 2015] dmar: IOMMU 1: reg_base_addr f7efe000 ver 1:0 cap
d2078c106f0466 ecap f020de
[Thu Dec 17 09:07:22 2015] dmar: DRHD base: 0x000000fbefe000 flags: 0x0
[Thu Dec 17 09:07:22 2015] dmar: IOMMU 2: reg_base_addr fbefe000 ver 1:0 cap
d2078c106f0466 ecap f020de
[Thu Dec 17 09:07:22 2015] dmar: DRHD base: 0x000000ecffe000 flags: 0x1
[Thu Dec 17 09:07:22 2015] dmar: IOMMU 3: reg_base_addr ecffe000 ver 1:0 cap
d2078c106f0466 ecap f020de
[Thu Dec 17 09:07:22 2015] dmar: RMRR base: 0x000000bdffd000 end: 0x000000bdffffff
......
[Thu Dec 17 09:07:22 2015] dmar: RMRR base: 0x000000bddde000 end: 0x000000bdddefff
[Thu Dec 17 09:07:22 2015] dmar: ATSR flags: 0x0
```

```
[Thu Dec 17 09:07:22 2015] IOAPIC id 12 under DRHD base  0xfbefe000 IOMMU 2
[Thu Dec 17 09:07:22 2015] IOAPIC id 11 under DRHD base  0xf7efe000 IOMMU 1
[Thu Dec 17 09:07:22 2015] IOAPIC id 10 under DRHD base  0xf34fe000 IOMMU 0
[Thu Dec 17 09:07:22 2015] IOAPIC id 8 under DRHD base  0xecffe000 IOMMU 3
[Thu Dec 17 09:07:22 2015] IOAPIC id 0 under DRHD base  0xecffe000 IOMMU 3
[Thu Dec 17 09:07:24 2015] IOMMU 2 0xfbefe000: using Queued invalidation
[Thu Dec 17 09:07:24 2015] IOMMU 1 0xf7efe000: using Queued invalidation
[Thu Dec 17 09:07:24 2015] IOMMU 0 0xf34fe000: using Queued invalidation
[Thu Dec 17 09:07:24 2015] IOMMU 3 0xecffe000: using Queued invalidation
[Thu Dec 17 09:07:24 2015] IOMMU: Setting RMRR:
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:01:00.0
[0xbddde000 - 0xbdddefff]
......
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:01:00.2
[0xbddde000 - 0xbdddefff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:01:00.4
[0xbddde000 - 0xbdddefff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:03:00.0
[0xe8000 - 0xe8fff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:03:00.1
[0xe8000 - 0xe8fff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:02:00.0
[0xe8000 - 0xe8fff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:02:00.1
[0xe8000 - 0xe8fff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:06:00.0
[0xe8000 - 0xe8fff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:06:00.1
[0xe8000 - 0xe8fff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:01:00.0
[0xe8000 - 0xe8fff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:01:00.2
[0xe8000 - 0xe8fff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:21:00.0
[0xe8000 - 0xe8fff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:21:00.1
[0xe8000 - 0xe8fff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:23:00.0
[0xe8000 - 0xe8fff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:23:00.1
[0xe8000 - 0xe8fff]
......
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:23:00.0
[0xbdf83000 - 0xbdf84fff]
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:23:00.1
[0xbdf83000 - 0xbdf84fff]
......
[Thu Dec 17 09:07:24 2015] IOMMU: Prepare 0-16MiB unity mapping for LPC
[Thu Dec 17 09:07:24 2015] IOMMU: Setting identity map for device 0000:00:1f.0 [0x0 -
0xffffff]
```

1.2.3. install required linux kernel for SR-IOV based VMX

linux kernel needs to be changed if VMX needs to be setup with `SR-IOV`.

currently the ixgbe coming with ubuntu does not work on VMX. The main issue is **lack of multicast support on ingress - packet received on a VF will be discarded siliently and won't be delivered into the guest VM, an example of the immediate effect of this is that OSPF (and most of today's IGP) neighborship won't come up**. Therefore building VMX based on `SR-IOV` requires to re-compile the ixgbe kernel driver from source code, which is provided by Juniper to fix the multicast support. the code is available in the installation package. At the time of the writing of this document there is problem to compile ixgbe from source code under any kernels other than `3.13.0-32-generic`. that's why the kernel needs to be changed in this setup.

There is a statement about this issue in the VMX document:

> Modified IXGBE drivers are included in the package. Multicast promiscuous mode for Virtual Functions is needed to receive control traffic that comes with broadcast MAC addresses. The reference driver does not come with this mode set, so the IXGBE drivers in this package contain certain modifications to overcome this limitation.
>
> — VMX "Getting Started Guide"

there is a plan to make ixgbe kernel module to work with newer kernel in new VMX release.

use below commands (provided in VMX installation doc) to change kernel:

```
sudo apt-get install linux-firmware linux-image-3.13.0.32-generic \
                     linux-image-extra-3.13.0.32-generic \
                     linux-headers-3.13.0.32-generic
```

setup default linux kernel

After changing the kernel for the next reboot, you may want to make it default kernel for later reboot. To achieve that following below steps:

in file `/boot/grub/grub.cfg` locate this line:

```
menuentry 'Ubuntu, with Linux 3.13.0-32-generic'
```

then move it before the first `menuentry` entry:

```
    export linux_gfx_mode
                                                #<------move to here
    menuentry 'Ubuntu' --class ubuntu -  ....
```

save and reboot the server.

this is optional because otherwise you will still be given a change to select kernel version during system reboot.

1.2.4. install required software packages

Use below commands (provided in VMX installation doc) to install required software packages:

```
sudo apt-get update
sudo apt-get install bridge-utils qemu-kvm libvirt-bin python numactl \
                     python-netifaces vnc4server libyaml-dev python-yaml\
                     libparted0-dev libpciaccess-dev libnuma-dev libyajl-dev\
                     libxml2-dev libglib2.0-dev libnl-dev libnl-dev python-pip\
                     python-dev libxml2-dev libxslt-dev
```

a quick way to verify if all/any of the required software package in the list were installed correctly or not, is to simply re-run the above installation commands again. If everything got installed correctly then you will get sth like this:

```
sudo apt-get install bridge-utils qemu-kvm libvirt-bin python numactl
\
python-netifaces vnc4server libyaml-dev python-yaml libparted0-dev \
libpciaccess-dev libnuma-dev libyajl-dev libxml2-dev libglib2.0-dev \
libnl-dev libnl-dev python-pip python-dev libxml2-dev libxslt-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'libxslt1-dev' instead of 'libxslt-dev'
bridge-utils is already the newest version.
libpciaccess-dev is already the newest version.
libxslt1-dev is already the newest version.
libyajl-dev is already the newest version.
python is already the newest version.
python-dev is already the newest version.
python-netifaces is already the newest version.
libnl-dev is already the newest version.
libglib2.0-dev is already the newest version.
libnuma-dev is already the newest version.
libparted0-dev is already the newest version.
libvirt-bin is already the newest version.
libxml2-dev is already the newest version.
libyaml-dev is already the newest version.
python-yaml is already the newest version.
qemu-kvm is already the newest version.
numactl is already the newest version.
python-pip is already the newest version.
vnc4server is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 124 not upgraded.
#<------
```

1.2.5. libvirt

make sure libvirt version 1.2.8 is installed for "performance version" of VMX. refer to the VMX document for detail steps to install it from source code.

In below command captures we demonstrate the libvirt upgrading process.

1. original libvirt coming with ubuntu14.02:

```
ping@ubuntu:~$ libvirtd --version
libvirtd (libvirt) 1.2.2
```

2. download and prepare source code:

```
cd /tmp
wget http://libvirt.org/sources/libvirt-1.2.8.tar.gz
tar zxvf libvirt-1.2.8.tar.gz
```

3. stop and uninstall old version:

```
cd libvirt-1.2.8
sudo ./configure --prefix=/usr/local --with-numactl

    checking for a BSD-compatible install... /usr/bin/install -c
    checking whether build environment is sane... yes
    checking for a thread-safe mkdir -p... /bin/mkdir -p
    checking for gawk... gawk
    checking whether make sets $(MAKE)... yes
    ......

sudo service libvirt-bin stop
    libvirt-bin stop/waiting

sudo make uninstall
    Making uninstall in .
    make[1]: Entering directory `/tmp/libvirt-1.2.8'
    make[1]: Leaving directory `/tmp/libvirt-1.2.8'
    ......

/bin/rm rf /usr/local/lib/libvirt*in/rm:
    cannot remove '/usr/local/lib/libvirt*': No such file or directory
```

4. install new version:

```
sudo ./configure --prefix=/usr --localstatedir=/ --with-numactl
    checking for a BSD-compatible install... /usr/bin/install -c
    checking whether build environment is sane... yes
    checking for a thread-safe mkdir -p... /bin/mkdir -p
......

sudo make
    make  all-recursive
    make[1]: Entering directory `/tmp/libvirt-1.2.8'
    Making all in .
    make[2]: Entering directory `/tmp/libvirt-1.2.8'
    make[2]: Leaving directory `/tmp/libvirt-1.2.8'
    Making all in gnulib/lib
    make[2]: Entering directory `/tmp/libvirt-1.2.8/gnulib/lib'
      GEN      alloca.h
      GEN      c++defs.h
      GEN      warn-on-use.h
      GEN      arg-nonnull.h
      GEN      arpa/inet.h
    ......

sudo make install
    Making install in .
    make[1]: Entering directory `/tmp/libvirt-1.2.8'
    make[2]: Entering directory `/tmp/libvirt-1.2.8'
    ......
```

5. start new version:

```
ping@ubuntu:/tmp/libvirt-1.2.8$ sudo service libvirt-bin start
libvirt-bin start/running, process 24450
ping@ubuntu:/tmp/libvirt-1.2.8$ ps aux| grep libvirt

ping@ubuntu:/tmp/libvirt-1.2.8$ ps aux| grep libvirt
root     24450  0.5  0.0 405252 10772 ?        Sl   21:40   0:00 /usr/sbin/libvirtd
-d
```

6. verify the new version:

```
ping@trinity:~$ libvirtd --version
libvirtd (libvirt) 1.2.8

ping@trinity:~$ service libvirt-bin status
libvirt-bin start/running, process 1559

ping@trinity:~$ which libvirtd
/usr/sbin/libvirtd

ping@trinity:~$ /usr/sbin/libvirtd --version
/usr/sbin/libvirtd (libvirt) 1.2.8

ping@trinity:~$ virsh --version
1.2.8

ping@trinity:/images/vmx_20151102.0$ sudo virsh -c qemu:///system version
Compiled against library: libvirt 1.2.8
Using library: libvirt 1.2.8
Using API: QEMU 1.2.8
Running hypervisor: QEMU 2.0.0
```

1.3. download vmx installation package

locate the vmx tarball

download VMX tarball from internal or public server.

- public URL
- internal server:

```
pings@svl-jtac-tool02:/volume/publish/dev/wrlinux/mx86/15.1F_att_drop$ ls -l
total 4180932
-rw-rw-r--  1 rbu-builder  rbu-builder   816737275 Nov  9 12:23 vmx_20151102.0.tgz
#<------
-rw-rw-r--  1 rbu-builder  rbu-builder  3447736320 Nov  4 12:18
vmx_20151102.0_tarball_issue.tgz
```

To have a quick look at the guest image files included in the tarball:

```
pings@svl-jtac-tool02:~$ cd /volume/publish/dev/wrlinux/mx86/15.1F_att_drop/
pings@svl-jtac-tool02:~$ tar tf vmx_20151102.0.tgz | grep images
vmx_20151102.0/images/
vmx_20151102.0/images/jinstall64-vmx-15.1F-20151104.0-domestic.img      ①
vmx_20151102.0/images/vFPC-20151102.img                                 ②
vmx_20151102.0/images/vmxhdd.img                                        ③
vmx_20151102.0/images/metadata_usb.img
```

① `jinstall` image, vRE/VCP (Virtual Control Plane) VM image

② `vFPC` image, vFPC/VFP (Virtual Forwarding Plane) VM image

③ `hdd` image, vRE virtual harddisk image

when downloaded from Juniper internal server, the `jinstall` image (vRE guest VM) may or may not be included in the VMX tarball. it is available in the normal Junos releases archives. tarball downloaded from public URL will always include `jinstall` and all other necessary images.

locate the vmx jinstall image

If the tarball does not contain a vRE jinstall image, we can download the image from other places and copy it over to the same "images" folder after untar the VMX tarball.

```
pings@svl-jtac-tool02:/volume/build/junos/15.1F/daily/20151102.0/ship$
ls -l | grep install64 | grep vmx
1 builder   748510583 jinstall64-vmx-15.1F-20151102.0-domestic-signed.tgz
1 builder          33 jinstall64-vmx-15.1F-20151102.0-domestic-signed.tgz.md5
1 builder          41 jinstall64-vmx-15.1F-20151102.0-domestic-signed.tgz.sha1
1 builder  1005715456 jinstall64-vmx-15.1F-20151102.0-domestic.img      #<----
1 builder          33 jinstall64-vmx-15.1F-20151102.0-domestic.img.md5
1 builder          41 jinstall64-vmx-15.1F-20151102.0-domestic.img.sha1
1 builder   748226059 jinstall64-vmx-15.1F-20151102.0-domestic.tgz
1 builder          33 jinstall64-vmx-15.1F-20151102.0-domestic.tgz.md5
1 builder          41 jinstall64-vmx-15.1F-20151102.0-domestic.tgz.sha1
```

There is no guarantee that an arbitrary combination of `jinstall` and `vFPC` images can work together or not. The publicly released VMX packages should already include the tested and working combination. Read the offical instruction and document coming with the software release before starting to install the VMX.

1.4. prepare a "work folder" for the installation

in my server I organize folder/files in this structure:

```
/virtualization             ①
 ├──── images                  ②
 │     ├──── ubuntu.img          ③
 │     ├──── vmx_20151102.0      ③
 │     │     ├──── build
 │     │     │     └──── vmx1
 │     │     │          ├──── images
 │     │     │          ├──── logs
 │     │     │          │     └──── vmx_1448293071.log
 │     │     │          └──── xml
 │     │     ├──── config
 │     │     │     ├──── samples
 │     │     │     │     ├──── vmx.conf.sriov
 │     │     │     │     ├──── vmx.conf.virtio
 │     │     │     │     └──── vmx-galaxy.conf
 │     │     │     ├──── vmx.conf            ⑥
 │     │     │     ├──── vmx.conf.sriov1     ⑥
 │     │     │     ├──── vmx.conf.sriov2     ⑥
 │     │     │     ├──── vmx.conf.ori
 │     │     │     └──── vmx-junosdev.conf
 │     │     ├──── docs

...<snipped>...

 │     └──── vmx_20151102.0.tgz  ③
 ├──── vmx1                    ④
 │     ├──── br-ext-generated.xml \
 │     ├──── br-int-generated.xml  |
 │     ├──── cpu_affinitize.sh     |     ⑤
 │     ├──── vfconfig-generated.sh |
 │     ├──── vPFE-generated.xml    |
 │     ├──── vRE-generated.xml    /
 │     └──── vmxhdd.img          ⑦
 └──── vmx2                    ④
       ├──── br-ext-generated.xml  \
       ├──── br-int-generated.xml   |
       ├──── cpu_affinitize.sh      |    ⑤
       ├──── vfconfig-generated.sh  |
       ├──── vPFE-generated.xml     |
       ├──── vRE-generated.xml     /
       └──── vmxhdd.img          ⑦

27 directories, 136 files
```

① parent folder to hold all virtualization files/releases/images

② "images" sub-folder holds all images for virtualizations

③ VM "images", installation files, tarballs, etc

④ sub-folder to hold all files for installation of multiple VMX VM instances

⑤ files generated by installation scripts

⑥ VMX installation configuration file

⑦ "hard disk" image file, holding all current VMX guest VM configs

Chapter 2. install VMX using installation script (SR-IOV)

Included in the tarball there is an orchestration script to automate the VMX setup in the server. running this script without any parameter will provide a quick help of the usage.

```
ping@trinity:/virtualization/images/vmx_20151102.0$ sudo ./vmx.sh

Usage: vmx.sh [CONTROL OPTIONS]
       vmx.sh [LOGGING OPTIONS] [CONTROL OPTIONS]
       vmx.sh [JUNOS-DEV BIND OPTIONS]
       vmx.sh [CONSOLE LOGIN OPTIONS]

    CONTROL OPTIONS:
       --install                     : Install And Start vMX
       --start                       : Start vMX
       --stop                        : Stop vMX
       --restart                     : Restart vMX
       --status                      : Check Status Of vMX
       --cleanup                     : Stop vMX And Cleanup Build Files
       --cfg <file>                  : Override With The Specified vmx.conf File
       --env <file>                  : Override With The Specified Environment .env
File
       --build <directory>           : Override With The Specified Directory for
Temporary Files
       --help                        : This Menu

    LOGGING OPTIONS:
       -l                            : Enable Logging
       -lv                           : Enable Verbose Logging
       -lvf                          : Enable Foreground Verbose Logging

    JUNOS-DEV BIND OPTIONS:
       --bind-dev                    : Bind Junos Devices
       --unbind-dev                  : Unbind Junos Devices
       --bind-check                  : Check Junos Device Bindings
       --cfg <file>                  : Override With The Specified vmx-junosdev.conf
File

    CONSOLE LOGIN OPTIONS:
       --console [vcp|vfp] [vmx_id]   : Login to the Console of VCP/VFP

    VFP Image OPTIONS:
       --vfp-info <VFP Image Path>    : Display Information About The Specified vFP
image

Copyright(c) Juniper Networks, 2015
```

2.1. the vmx.conf file

The config file `vmx.conf` is a centralized place where all of the installation parameters and options are defined. The installation script `vmx.sh` scan this file as input and generate XML files and shell scripts as output. the generated XML files will be read by libvirt/virsh tool later as input information to setup and manipulate the VMs.

The generated shell script will be executed to configure:

- vcpu pinning (both SRIOV and VIRTIO)
- VF properties (SRIOV only)

For better readability `vmx.conf` is implemented in YAML format. However, in VMX the guest VM instances are manipulated via libvirt, which currently solely relies on XML. This is why there are 2 software modules in the pre-installed packages to support YAML to XML conversion:

- libyaml-dev Fast YAML 1.1 parser and emitter library (development)
- python-yaml YAML parser and emitter for Python

a short introduction about YAML

YAML (YAML Ain't Markup) is a human friendly data serialization language. VMX config file uses YAML because it's as close to plain English as data serialization and configuration formats get. The advantage of YAML is that it does not require curly braces, allowing you to omit quotation marks for strings in most cases, **relying on indentation** for structure, which makes it much more readable compared to XML.

Important tips about YAML:

- indentation matters: YAML relies on indentation to understand the data structure,
- use `space` instead of `tabs`, `tabs` are not universally supported across implementations
- case sensitive

In short, every space/indentation matters. Taking cautions when modifying the `vmx.conf` file. A good practice is to just replace the parameters (numbers, image path, interface names, MAC, etc) and leave everything else intact.

`vmx.conf` template coming with the VMX tarball looks :files

```
ping@trinity:/virtualization/images/vmx_20151102.0/config$ cat vmx.conf
#############################################################
#
#  vmx.conf
#  Config file for vmx on the hypervisor.
```

```
#  Uses YAML syntax.
#  Leave a space after ":" to specify the parameter value.
#
#################################################################

---
#Configuration on the host side - management interface, VM images etc.
HOST:                                                                    ①
    identifier                : vmx1   # Maximum 4 characters            ②
    host-management-interface : eth0                                     ③
    routing-engine-image      : "/home/vmx/vmxlite/images/jinstall64-vmx.img" ④
    routing-engine-hdd        : "/home/vmx/vmxlite/images/vmxhdd.img"    ④
    forwarding-engine-image   : "/home/vmx/vmxlite/images/vPFE.img"      ④

---
#External bridge configuration                                           ⑤
BRIDGES:
    - type  : external
      name  : br-ext                 # Max 10 characters

---
#vRE VM parameters                                                       ⑥
CONTROL_PLANE:
    vcpus       : 1
    memory-mb   : 1024
    console_port: 8601

    interfaces  :
      - type      : static
        ipaddr    : 10.102.144.94
        macaddr   : "0A:00:DD:C0:DE:0E"

---
#vPFE VM parameters                                                      ⑦
FORWARDING_PLANE:
    memory-mb   : 6144
    vcpus       : 3
    console_port: 8602
    device-type : virtio                                                 ⑧

    interfaces  :
      - type      : static
        ipaddr    : 10.102.144.98
        macaddr   : "0A:00:DD:C0:DE:10"

---
#Interfaces                                                              ⑨
JUNOS_DEVICES:
   - interface           : ge-0/0/0
     mac-address         : "02:06:0A:0E:FF:F0"
     description         : "ge-0/0/0 interface"
```

```
  - interface           : ge-0/0/1
    mac-address         : "02:06:0A:0E:FF:F1"
    description         : "ge-0/0/0 interface"

  - interface           : ge-0/0/2
    mac-address         : "02:06:0A:0E:FF:F2"
    description         : "ge-0/0/0 interface"

  - interface           : ge-0/0/3
    mac-address         : "02:06:0A:0E:FF:F3"
    description         : "ge-0/0/0 interface"
```

① "HOST" config section.

② ID of the vmx instance. this string will be encoded into the final VM name: vcp-<ID> or vfp-<ID> .

③ current management interface of the server. the installation script will "move" the IP/MAC property from this port to an external bridge named "br-ext".

④ vRE/vFPC/Harddisk VM images location.

⑤ external bridge configuration section: a bridge utility, named `br-ext`, will be created, for managment connection from/to the external networks.

⑥ vRE configuration section: this template uses 1 vCPU, 1G mem, console port 8601 to start vRE guest VM. the VM mgmt interface's "peer interface" [1: interface showing in host, and peering with guest VM interface, e.g. the `vcp_ext-vmx1` interface is "peer interface" of fxp0 in VMX]from the host - `vcp_ext-vmx1` ("attached" to `fxp0` port from inside the guest VM), will be configured with the specified MAC address. the corresponding `fxp0` interface from inside of vRE VM will inherit same MAC from it. [2: the IP address specified here still needs to be configured manually from inside of the vRE guest VM, unless dhcp client is implemented from guest VM to request for an IP from the DHCP server running in host OS, this may be implemented in the future releases]

⑦ vFPC configuration section: this template uses 3 vCPU, 6G mem, console port 8602 to build vFPC guest VM. the VM mgmt interface's "peer interface" [1: interface showing in host, and peering with guest VM interface, e.g. the `vcp_ext-vmx1` interface is "peer interface" of fxp0 in VMX] from the host - `vfp_ext-vmx1` ("attached" to `ext` port from side the guest VM) will be configured with the specified MAC address. the `ext` interface from inside the vFPC guest VM will inherit same MAC from it.

⑧ as a KVM implementation, VMX currently supports two type of network IO virtualization : `VT-d` + `SRIOV`, or `VIRTIO`. this config knob `device-type` will determine which IO virtualization technology will be used to build VMX. This template uses "virtio" IO virtualization.

⑨ VMX router interface configuration section: This is where the router `ge-0/0/z` properties can be configured. depending on `device-type` value the available configurable properties will be different. Since this template uses "virtio" virtualization, only "mac-address" is configurable. more details will be covered in later sections of this doc.

2.2. assigning MAC address

the "virtual" MX will use virual NIC running inside the guest VM. so unlike any real NIC coming

with a built-in MAC address which is globally uniquely assigned, the MAC address of "vNIC" is what you assigned before or after the VMX instances were brought up.

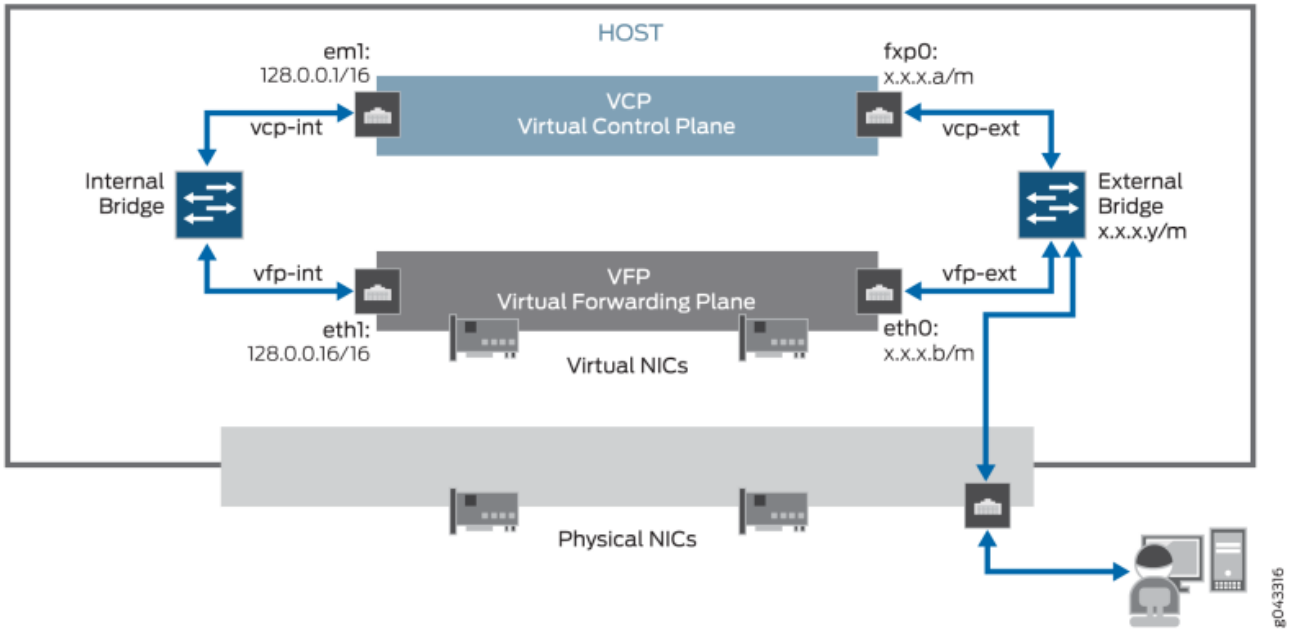


Figure 1. internal and external interfaces of VFP and VCP VM

To avoid confliction to other external devices, at least these below MAC addresses needs to be unique in the diagram:

1. MAC for ge-0/0/z
2. fxp0 on VRE(VCP)
3. eth1 or ext on VPFE(VFP)

VFP VM interface name changed in new VMX release.

Table 2. VFP interface name

before 15.1	**from 15.1**	**roll**
eth1	ext	VFP interface facing external networks
eth0	int	VFP interface for internal use

These MAC addresses will exit the server and be learnt by the external device.

Below are internal / isolated interfaces which never communicate with external devices, MAC address doesn't matter for these interfaces:

1. em1 on VRE
2. eth0 or int on VPFE

"Locally Administered MAC Address" is good candidate to be used in lab test environment:

```
x2-xx-xx-xx-xx-xx
x6-xx-xx-xx-xx-xx
xA-xx-xx-xx-xx-xx
xE-xx-xx-xx-xx-xx
```

In my setup I follow this simply rule below to avoid MAC address confliction with other systems in the lab network:

```
mac-address          : "02:04:17:01:02:02"
                         - ----- -- -- --
                         |    |   |  |  |
                         |    |   |  | ⑤
                         |    |   | ④
                         |    |  ③
                         |   ②
                        ①
```

① locally administered MAC address

② last 4 digits of IP (e.g. x.y.4.17) or MAC of management interface.

③ VMX instance number, first VMX instance uses `01`, second uses `02`, etc.

④ `01` for control plane interface (`fxp0` for VCP, `ext` for VFP) of a VM, `02` for forwarding plane interface (ge-0/0/z) of a VM

⑤ assign a unique number for each type of interface

Table 3. an example of MAC address allocation with this rule

VMX instance1	vRE fxp0	02:04:17:01:01:01
	vFPC ext	02:04:17:01:01:02
	ge-0/0/0	02:04:17:01:02:01
	ge-0/0/1	02:04:17:01:02:02
VMX instance2	vRE fxp0	02:04:17:02:01:01
	vFPC ext	02:04:17:02:01:02
	ge-0/0/0	02:04:17:02:02:01
	ge-0/0/1	02:04:17:02:02:02

2.3. modify the vmx.conf (SR-IOV)

The config file needs to be changed according to the installation plan. this is vmx.conf file that I use to setup SR-IOV based VMX.

```
ping@trinity:/virtualization/images/vmx_20151102.0/config$ cat vmx.conf
################################################################
```

```
#
#  vmx.conf
#  Config file for vmx on the hypervisor.
#  Uses YAML syntax.
#  Leave a space after ":" to specify the parameter value.
#
#####################################################################

---
#Configuration on the host side - management interface, VM images etc.
HOST:
    identifier                : vmx1   # Maximum 4 characters
    host-management-interface : em1
    routing-engine-image      :
"/virtualization/images/vmx_20151102.0/images/jinstall64-vmx-15.1F-20151104.0-
domestic.img"
    routing-engine-hdd        : "
/virtualization/images/vmx_20151102.0/vmx1/vmxhdd.img"
    forwarding-engine-image   : "/virtualization/images/vmx_20151102.0/images/vFPC-
20151102.img"

---
#External bridge configuration
BRIDGES:
    - type  : external
      name  : br-ext                  # Max 10 characters

---
#vRE VM parameters
CONTROL_PLANE:
    vcpus       : 1
    memory-mb   : 2048
    console_port: 8816

    interfaces  :
      - type      : static
        ipaddr    : 10.85.4.105
        macaddr   : "02:04:17:01:01:01"
---
#vPFE VM parameters
FORWARDING_PLANE:
    memory-mb   : 16384
    vcpus       : 4                     ①
    console_port: 8817
    device-type : sriov

    interfaces  :
      - type      : static
        ipaddr    : 10.85.4.106
        macaddr   : "02:04:17:01:01:02"
```

```
---
#Interfaces
JUNOS_DEVICES:
    - interface            : ge-0/0/0
      port-speed-mbps      : 10000      ②
      nic                  : p3p1       ②
      mtu                  : 2000             # DO NOT EDIT ②
      virtual-function     : 0          ②
      mac-address          : "02:04.17:01:02:01"
      description          : "ge-0/0/0 connects to eth6"

    - interface            : ge-0/0/1
      port-speed-mbps      : 10000      ②
      nic                  : p2p1       ②
      mtu                  : 2000             # DO NOT EDIT
      virtual-function     : 0          ②
      mac-address          : "02:04.17:01:02:02"
      description          : "ge-0/0/1 connects to eth7"
```

① assign 4 CPUs to vPFE VM

② SR-IOV only options

key parameters in this conf file:

- my server's mgmt interface name is `em1`
- vRE and vFPC images location will be just the images folder from the untar.ed installation package. these images can then be shared by all VMX instances.
- virtual harddisk image `vmxhdd.img` will be in a seperate folder created specifically for current VMX instance
- use console port 88x6 for vRE guest VM and 88x7 for vFPC guest VM, where x = instance number. Any other number which is not yet in use is fine.
- MAC addresses are allocated following the above mentioned rule
- for SR-IOV virtualization, these link properties need to be configured:
  - physical NIC name
  - VF number
  - MAC address
  - link speed
  - MTU

for virtio virtualization, only MAC address can be defined in the config file. the MTU can be defined in a seperate file `vmx-junosdev.conf`.

In this example only 4 CPUs were assigned to vPFE VM, which is less than what is required for full performance in production environment ("performance mode").

In VMX 15.1, recommended number of CPU for "performance mode" are calculated as following:

- 1 for host-if
- 1 for flow-manager
- 1 for vmxt process
- 2 per each "IO thread", 1 for each receiving and 1 for sending
- remaining for "worker thread"

non-performance mode or "lite" mode requires less CPU - minimum 3 CPU is fine to bring up the VFP.

Here in lab environment, for test/study purpose I was able to bring up 2 interfaces SR-IOV VMX with totally 5 CPUs - 1 for VCP and 4 for VFP.

2.4. run the installation script: SR-IOV

After modification of vmx.conf file, we can run the `vmx.sh` script to setup the VMX.

```
ping@trinity:/virtualization/images/vmx_20151102.0$ sudo ./vmx.sh --install
==============================================
    Welcome to VMX
==============================================
Date.....................................11/24/15 21:18:36
VMX Identifier.....................................vmx1
Config
file......................................./virtualization/images/vmx_20151102.0/confi
g/vmx.conf
Build
Directory......................................./virtualization/images/vmx_20151102.0/buil
d/vmx1
Environment
file................................/virtualization/images/vmx_20151102.0/env/ubuntu
_sriov.env
Junos Device Type.................................sriov
Initialize scripts................................[OK]
Copy images to build directory....................[OK]
==============================================
    VMX Environment Setup Completed
==============================================
==============================================
    VMX Install & Start
==============================================
Linux distribution...............................ubuntu
```

```
Intel IOMMU status.............................[Enabled]
Verify if GRUB needs reboot....................[No]
Installation status of qemu-kvm................[OK]
Installation status of libvirt-bin.............[OK]
Installation status of bridge-utils............[OK]
Installation status of python..................[OK]
Installation status of libyaml-dev.............[OK]
Installation status of python-yaml.............[OK]
Installation status of numactl.................[OK]
Installation status of libnuma-dev.............[OK]
Installation status of libparted0-dev..........[OK]
Installation status of libpciaccess-dev........[OK]
Installation status of libyajl-dev.............[OK]
Installation status of libxml2-dev.............[OK]
Installation status of libglib2.0-dev..........[OK]
Installation status of libnl-dev...............[OK]
Check Kernel version...........................[OK]
Check Qemu version.............................[OK]
Check libvirt version..........................[OK]
Check virsh connectivity.......................[OK]
Check IXGBE drivers............................[OK]
================================================
    Pre-Install Checks Completed
================================================
Check for VM vcp-vmx1..........................[Running]
Shutdown vcp-vmx1..............................[OK]
Check for VM vfp-vmx1..........................[Running]
Shutdown vfp-vmx1..............................[OK]
Cleanup VM states..............................[OK]
Check if bridge br-ext exists..................[Yes]
Get Configured Management Interface............em1
Find existing management gateway...............br-ext
Mgmt interface needs reconfiguration...........[Yes]
Gateway interface needs change.................[Yes]
Check if br-ext has valid IP address...........[Yes]
Get Management Address.........................10.85.4.17
Get Management Mask............................255.255.255.128
Get Management Gateway.........................10.85.4.1
Del em1 from br-ext............................[OK]
Configure em1..................................[Yes]
Cleanup VM bridge br-ext.......................[OK]
Cleanup VM bridge br-int-vmx1..................[OK]
Cleanup IXGBE drivers..........................[OK]
================================================
    VMX Stop Completed
================================================
Check VCP image................................[OK]
Check VFP image................................[OK]
VMX Model......................................FPC
Check VCP Config image.........................[OK]
Check management interface.....................[OK]
```

```
Check interface p3p1.............................[OK]
Check interface p2p1.............................[OK]
Setup huge pages to 32768........................[OK]
Number of Intel 82599 NICs.......................8
Configuring Intel 82599 Adapters for SRIOV........[OK]
Number of Virtual Functions created..............[OK]
Attempt to kill libvirt..........................[OK]
Attempt to start libvirt.........................[OK]
Sleep 2 secs.....................................[OK]
Check libvirt support for hugepages..............[OK]
================================================
    System Setup Completed
================================================
Get Management Address of em1....................[OK]
Generate libvirt files...........................[OK]
Sleep 2 secs.....................................[OK]
Configure virtual functions for SRIOV............[OK]
Find configured management interface.............em1
Find existing management gateway.................em1
Check if em1 is already enslaved to br-ext........[No]
Gateway interface needs change...................[Yes]
Create br-ext....................................[OK]
Get Management Gateway...........................10.85.4.1
Flush em1........................................[OK]
Start br-ext.....................................[OK]
Bind em1 to br-ext...............................[OK]
Get Management MAC...............................38:ea:a7:37:7c:54
Assign Management MAC 38:ea:a7:37:7c:54..........[OK]
Add default gw 10.85.4.1.........................[OK]
Create br-int-vmx1...............................[OK]
Start br-int-vmx1................................[OK]
Check and start default bridge...................[OK]
Define vcp-vmx1..................................[OK]
Define vfp-vmx1..................................[OK]
Wait 2 secs......................................[OK]
Start vcp-vmx1...................................[OK]
Start vfp-vmx1...................................[OK]
Wait 2 secs......................................[OK]
Perform CPU pinning..............................[OK]
================================================
    VMX Bringup Completed
================================================
Check if br-ext is created.......................[Created]
Check if br-int-vmx1 is created..................[Created]
Check if VM vcp-vmx1 is running..................[Running]
Check if VM vfp-vmx1 is running..................[Running]
Check if tap interface vcp_ext-vmx1 exists........[OK]
Check if tap interface vcp_int-vmx1 exists........[OK]
Check if tap interface vfp_ext-vmx1 exists........[OK]
Check if tap interface vfp_int-vmx1 exists........[OK]
================================================
```

```
    VMX Status Verification Completed.
================================================
Log file......................................./dev/null
================================================
    Thankyou for using VMX
================================================
```

2.5. quick verification

After the installation we need to know if the installed VMX is working well.

1. list running VMs: vRE and vFPC

```
ping@trinity:~$ sudo virsh list
[sudo] password for ping:
 Id    Name                           State
----------------------------------------------------
 2     vcp-vmx1                       running
 3     vfp-vmx1                       running
```

2. login to vRE

```
ping@trinity:~$ telnet localhost 8816
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Amnesiac (ttyd0)

login: root
root@% cli
root>
```

3. verify if virtual PFE is "online"

```
labroot> show chassis fpc
                     Temp  CPU Utilization (%)   CPU Utilization (%)  Memory
Utilization (%)
Slot State            (C)  Total  Interrupt      1min   5min   15min  DRAM (MB)
Heap     Buffer
  0  Online           Testing   3         0        3      3      2      1         6
0

labroot> show chassis fpc pic-status
Slot 0   Online       Virtual FPC
  PIC 0  Online       Virtual
```

This may take a couple of minutes.

4. login to vPFE

```
root@trinity:~# telnet localhost 8817
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.


Wind River Linux 6.0.0.12 vfp-vmx1 console

vfp-vmx1 login: pfe
Password:
pfe@vfp-vmx1:~$ cat /etc/issue.net
Wind River Linux 6.0.0.12 %h
```

there are two built-in account to login vfp yocto VM:

- `pfe` with password `pfe`
- root account is `root` with password `root`.

Later we'll demonstrate ping and OSPF neighborship test. A more intensive verification requires to enable more features and protocols (OSPF/BGP/multicast/etc), which is beyond the scope of this doc.

2.6. uninstall (cleanup) VMX with installation script

To uninstall VMX just run the same script with `--cleanup` option:

```
ping@trinity:/virtualization/images/vmx_20151102.0$ sudo ./vmx.sh --cleanup
================================================
    Welcome to VMX
================================================
```

```
Date.....................................11/24/15 21:25:41
VMX Identifier.....................................vmx1
Config
file...................................../virtualization/images/vmx_20151102.0/confi
g/vmx.conf
Build
Directory.................................../virtualization/images/vmx_20151102.0/buil
d/vmx1
Environment
file................................/virtualization/images/vmx_20151102.0/env/ubuntu
_sriov.env
Junos Device Type.................................sriov
Initialize scripts................................[OK]
================================================
    VMX Environment Setup Completed
================================================
================================================
    VMX Stop & Cleanup
================================================
Check if vMX is running..........................[Yes]
Check for VM vcp-vmx1............................[Running]
Shutdown vcp-vmx1................................[OK]
Check for VM vfp-vmx1............................[Running]
Shutdown vfp-vmx1................................[OK]
Cleanup VM states................................[OK]
Check if bridge br-ext exists....................[Yes]
Get Configured Management Interface..............em1
Find existing management gateway..................br-ext
Mgmt interface needs reconfiguration.............[Yes]
Gateway interface needs change...................[Yes]
Check if br-ext has valid IP address.............[Yes]
Get Management Address...........................10.85.4.17
Get Management Mask..............................255.255.255.128
Get Management Gateway...........................10.85.4.1
Del em1 from br-ext..............................[OK]
Configure em1....................................[Yes]
Cleanup VM bridge br-ext.........................[OK]
Cleanup VM bridge br-int-vmx1....................[OK]
Cleanup IXGBE drivers............................[OK]
================================================
    VMX Stop Completed
================================================
Cleanup auto-generated files.....................[OK]
================================================
    VMX Cleanup Completed
================================================
Log file.........................................../dev/null
================================================
    Thankyou for using VMX
================================================
```

If a different vmx config file was specified when installing VMX, specify the same as parameter of the script to clean up VMX.

```
ping@trinity:/virtualization/images/vmx_20151102.0$ sudo ./vmx.sh --install --cfg
config/vmx.conf.sriov.1
ping@trinity:/virtualization/images/vmx_20151102.0$ sudo ./vmx.sh --cleanup --cfg
config/vmx.conf.sriov.1
```

Chapter 3. install VMX using installation script (VIRTIO)

3.1. modify the vmx.conf (virtio)

Most configuration parameters in `vmx.conf` config file to setup `VIRTIO` version of VMX are the same as the one used to setup `SRIOV` VMX as shown above, a few differences are:

- for virtio virtualization, only MAC address can be defined in this config file
- MTU can to be configured in a seperate `junosdev.conf` config file
- `virtio` technology generates seperated virtual NICs that are used to build VMX; all L1 properties remains in the physical NIC. therefore no need to specify L1 properties like physical NIC, VF, link speed, etc for VIRTIO setup.
- To communicate with external networks, virtio virtual NIC can be "bound" to a physical NIC port, or another virtual NIC, using any existing technologies like linux bridge, OVS, etc.

```
ping@trinity:/virtualization/images/vmx_20151102.0/config$ cat vmx.conf.virtio.1
######################################################################
#
#  vmx.conf
#  Config file for vmx on the hypervisor.
#  Uses YAML syntax.
#  Leave a space after ":" to specify the parameter value.
#
######################################################################

---
#Configuration on the host side - management interface, VM images etc.
HOST:
    identifier                : vmx1   # Maximum 4 characters
    host-management-interface : em1
    routing-engine-image      :
"/virtualization/images/vmx_20151102.0/images/jinstall64-vmx-15.1F-20151104.0-
domestic.img"
    routing-engine-hdd        : "
/virtualization/images/vmx_20151102.0/vmx1/vmxhdd.img"
    forwarding-engine-image   : "/virtualization/images/vmx_20151102.0/images/vFPC-
20151102.img"

---
#External bridge configuration
BRIDGES:
    - type  : external
      name  : br-ext                  # Max 10 characters

---
```

```
#vRE VM parameters
CONTROL_PLANE:
    vcpus       : 1
    memory-mb   : 2048
    console_port: 8816

    interfaces  :
      - type      : static
        ipaddr    : 10.85.4.105
        macaddr   : "02:04:17:01:01:01"
        #macaddr   : "0A:00:DD:C0:DE:0E"
---
#vPFE VM parameters
FORWARDING_PLANE:
    memory-mb   : 4096
    vcpus       : 3
    console_port: 8817
    device-type : virtio

    interfaces  :
      - type      : static
        ipaddr    : 10.85.4.106
        macaddr   : "02:04:17:01:01:02"
        #macaddr   : "0A:00:DD:C0:DE:10"

---
#Interfaces
JUNOS_DEVICES:
    - interface           : ge-0/0/0
      mac-address         : "02:04:17:01:02:01"
      description         : "ge-0/0/0 connects to eth6"

    - interface           : ge-0/0/1
      mac-address         : "02:04:17:01:02:02"
      description         : "ge-0/0/1 connects to eth7"
```

3.2. run the installation script: virtio

After modification of vmx config file, we can run the same `vmx.sh` script to setup the VMX. This time we use another option: `--cfg`, to tell the script where the configuration file can be found. This is necessary if we defined a seperate config file other than the default `config/vmx.conf`.

```
ping@trinity:/virtualization/images/vmx_20151102.0$ sudo ./vmx.sh --install --cfg
config/vmx.conf.virtio.1
----------------------------------------------
    Welcome to VMX
----------------------------------------------
Date.....................................11/30/15 21:40:16
VMX Identifier.....................................vmx1
```

```
Config file.....................................
    /virtualization/images/vmx_20151102.0/config/vmx.conf.virtio.1
Build
Directory.................................../virtualization/images/vmx_20151102.0/buil
d/vmx1
Environment
file.............................../virtualization/images/vmx_20151102.0/env/ubuntu
_virtio.env
Junos Device Type...............................virtio
Initialize scripts..............................[OK]
Copy images to build directory..................[OK]
================================================
    VMX Environment Setup Completed
================================================
================================================
    VMX Install & Start
================================================
Linux distribution..............................ubuntu
Check GRUB......................................[Disabled]
Installation status of qemu-kvm.................[OK]
Installation status of libvirt-bin..............[OK]
Installation status of bridge-utils.............[OK]
Installation status of python...................[OK]
Installation status of libyaml-dev..............[OK]
Installation status of python-yaml..............[OK]
Installation status of numactl..................[OK]
Installation status of libnuma-dev..............[OK]
Installation status of libparted0-dev...........[OK]
Installation status of libpciaccess-dev.........[OK]
Installation status of libyajl-dev..............[OK]
Installation status of libxml2-dev..............[OK]
Installation status of libglib2.0-dev...........[OK]
Installation status of libnl-dev................[OK]
Check Kernel Version............................[Disabled]
Check Qemu Version..............................[Disabled]
Check libvirt Version...........................[Disabled]
Check virsh connectivity........................[OK]
IXGBE Enabled...................................[Disabled]
================================================
    Pre-Install Checks Completed
================================================
Check for VM vcp-vmx1...........................[Not Running]
Check for VM vfp-vmx1...........................[Not Running]
Cleanup VM states...............................[OK]
Check if bridge br-ext exists...................[No]
Cleanup VM bridge br-ext........................[OK]
Cleanup VM bridge br-int-vmx1...................[OK]
================================================
    VMX Stop Completed
================================================
Check VCP image.................................[OK]
```

```
Check VFP image.................................[OK]
VMX Model.....................................FPC
Check VCP Config image..........................[OK]
Check management interface......................[OK]
Setup huge pages to 32768.......................[OK]
Attempt to kill libvirt.........................[OK]
Attempt to start libvirt........................[OK]
Sleep 2 secs....................................[OK]
Check libvirt support for hugepages.............[OK]
================================================
    System Setup Completed
================================================
Get Management Address of em1...................[OK]
Generate libvirt files..........................[OK]
Sleep 2 secs....................................[OK]
Find configured management interface.............em1
Find existing management gateway.................em1
Check if em1 is already enslaved to br-ext.....[No]
Gateway interface needs change..................[Yes]
Create br-ext...................................[OK]
Get Management Gateway..........................10.85.4.1
Flush em1.......................................[OK]
Start br-ext....................................[OK]
Bind em1 to br-ext..............................[OK]
Get Management MAC..............................38:ea:a7:37:7c:54
Assign Management MAC 38:ea:a7:37:7c:54.........[OK]
Add default gw 10.85.4.1........................[OK]
Create br-int-vmx1..............................[OK]
Start br-int-vmx1...............................[OK]
Check and start default bridge..................[OK]
Define vcp-vmx1.................................[OK]
Define vfp-vmx1.................................[OK]
Wait 2 secs.....................................[OK]
Start vcp-vmx1..................................[OK]
Start vfp-vmx1..................................[OK]
Wait 2 secs.....................................[OK]
================================================
    VMX Bringup Completed
================================================
Check if br-ext is created.....................[Created]
Check if br-int-vmx1 is created................[Created]
Check if VM vcp-vmx1 is running................[Running]
Check if VM vfp-vmx1 is running................[Running]
Check if tap interface vcp_ext-vmx1 exists.....[OK]
Check if tap interface vcp_int-vmx1 exists.....[OK]
Check if tap interface vfp_ext-vmx1 exists.....[OK]
Check if tap interface vfp_int-vmx1 exists.....[OK]
================================================
    VMX Status Verification Completed.
================================================
Log file........................................./dev/null
```

```
================================================
    Thankyou for using VMX
================================================
```

3.3. uninstall (cleanup) VMX with installation script

To uninstall VMX just run the same script with and `--cleanup` option. again we use `--cfg` option to specify the `vmx.conf` file which we used to set up the VMX :

```
ping@trinity:/virtualization/images/vmx_20151102.0$ sudo ./vmx.sh --cleanup --cfg
config/vmx.conf.virtio.1
================================================
    Welcome to VMX
================================================
Date.....................................11/30/15 21:14:08
VMX Identifier.....................................vmx1
Config file.....................................
    /virtualization/images/vmx_20151102.0/config/vmx.conf.virtio.1
Build
Directory...................................../virtualization/images/vmx_20151102.0/buil
d/vmx1
Environment
file................................/virtualization/images/vmx_20151102.0/env/ubuntu
_virtio.env
Junos Device Type..............................virtio
Initialize scripts.............................[OK]
================================================
    VMX Environment Setup Completed
================================================
================================================
    VMX Stop & Cleanup
================================================
Check if vMX is running.........................[Yes]
Check for VM vcp-vmx1...........................[Running]
Shutdown vcp-vmx1...............................[OK]
Check for VM vfp-vmx1...........................[Running]
Shutdown vfp-vmx1...............................[OK]
Cleanup VM states...............................[OK]
Check if bridge br-ext exists...................[Yes]
Get Configured Management Interface.............em1
Find existing management gateway................br-ext
Mgmt interface needs reconfiguration............[Yes]
Gateway interface needs change..................[Yes]
Check if br-ext has valid IP address............[Yes]
Get Management Address..........................10.85.4.17
Get Management Mask.............................255.255.255.128
Get Management Gateway..........................10.85.4.1
Del em1 from br-ext.............................[OK]
Configure em1...................................[Yes]
```

```
Cleanup VM bridge br-ext.........................[OK]
Cleanup VM bridge br-int-vmx1....................[OK]
================================================
    VMX Stop Completed
================================================
Cleanup auto-generated files.....................[OK]
================================================
    VMX Cleanup Completed
================================================
Log file.........................................../dev/null
================================================
    Thankyou for using VMX
================================================
```

Chapter 4. setup multiple VMX instances

Building multiple VMX instances in a server is no much different than building a one instance VMX - Just running the installation script multiple time with a different config file each time will be almost sufficient.

Depending on the different release, The installation script may or may not has issue to build multiple instances in one server. There are always works in progress to improve the script (based on the feedbacks). In case it doesn't work well (e.g bailing out in the middle), you can always install VMX manually In this example we use `virtio` to demonstrate the process, `SRIOV` will be exactly the same process and the only difference is that more physical NIC or VFs are needed.

In this section we demonstrate how to set up 2 VIRTIO-based VMX instances, setting up multiple SRIOV-based VMX instances will be a same process.

4.1. config file for the first VMX instance

```
ping@trinity:/virtualization/images/vmx_20151102.0$ cat config/vmx.conf.virtio.1
######################################################################
#
#  vmx.conf
#  Config file for vmx on the hypervisor.
#  Uses YAML syntax.
#  Leave a space after ":" to specify the parameter value.
#
######################################################################

---
#Configuration on the host side - management interface, VM images etc.
HOST:
    identifier                : vmx1   # Maximum 4 characters
    host-management-interface : em1
    routing-engine-image      :
"/virtualization/images/vmx_20151102.0/images/jinstall64-vmx-15.1F-20151104.0-
domestic.img"
    routing-engine-hdd        :
"/virtualization/images/vmx_20151102.0/images/vmxhdd.img"
    forwarding-engine-image   : "/virtualization/images/vmx_20151102.0/images/vFPC-
20151102.img"

---
#External bridge configuration
BRIDGES:
    - type  : external
      name  : br-ext                # Max 10 characters

---
```

```
#vRE VM parameters
CONTROL_PLANE:
    vcpus       : 1
    memory-mb   : 2048
    console_port: 8816

    interfaces  :
      - type      : static
        ipaddr    : 10.85.4.105
        macaddr   : "02:04:17:01:01:01"
        #macaddr   : "0A:00:DD:C0:DE:0E"
---
#vPFE VM parameters
FORWARDING_PLANE:
    memory-mb   : 4096
    vcpus       : 3
    console_port: 8817
    device-type : virtio

    interfaces  :
      - type      : static
        ipaddr    : 10.85.4.106
        macaddr   : "02:04:17:01:01:02"
        #macaddr   : "0A:00:DD:C0:DE:10"

---
#Interfaces
JUNOS_DEVICES:
    - interface           : ge-0/0/0
      mac-address         : "02:04:17:01:02:01"
      description         : "ge-0/0/0 connects to eth6"

    - interface           : ge-0/0/1
      mac-address         : "02:04:17:01:02:02"
      description         : "ge-0/0/1 connects to eth7"
```

4.2. config file for the second VMX instance

Following parameters need to be modified before running the script again - this is to avoid confliction with the first instance.

- `HOST` section: `identifier`, the script will create a folder based on this string, VM "domain name" will also be based on this value.
- `console_port` in both `CONTROL_PLANE` and `FORWARDING_PLANE` section
- `macaddr` in both `CONTROL_PLANE` and `FORWARDING_PLANE` section: these are for VM mgmt port
- `JUNOS_DEVICES` section: settings for the Junos ge-0/0/x interface [1: interface showing in host, and peering with guest VM interface, e.g. the `vcp_ext-vmx1` interface is "peer interface" of fxp0 in VMX]:

- for VIRTIO-based VMX:
  1. `interface`
  2. `mac-address`:
- for SRIOV-based VMX, these extra settings are also needed to be configured:
  1. physical NIC name: `nic`
  2. `virtual-function`
  3. `port-speed-mbps`
  4. `mtu`

```
ping@trinity:/virtualization/images/vmx_20151102.0$ cat config/vmx.conf.virtio.2
######################################################################
#
#  vmx.conf
#  Config file for vmx on the hypervisor.
#  Uses YAML syntax.
#  Leave a space after ":" to specify the parameter value.
#
######################################################################

---
#Configuration on the host side - management interface, VM images etc.
HOST:
    identifier                : vmx2   # Maximum 4 characters
    host-management-interface : em1
    routing-engine-image      :
"/virtualization/images/vmx_20151102.0/images/jinstall64-vmx-15.1F-20151104.0-
domestic.img"
    routing-engine-hdd        :
"/virtualization/images/vmx_20151102.0/images/vmxhdd.img"
    forwarding-engine-image   : "/virtualization/images/vmx_20151102.0/images/vFPC-
20151102.img"

---
#External bridge configuration
BRIDGES:
    - type  : external
      name  : br-ext                  # Max 10 characters

---
#vRE VM parameters
CONTROL_PLANE:
    vcpus       : 1
    memory-mb   : 2048
    console_port: 8826

    interfaces  :
      - type      : static
```

```
        ipaddr    : 10.85.4.107
        macaddr   : "02:04:17:02:01:01"
        #macaddr   : "0A:00:DD:C0:DE:0E"
---
#vPFE VM parameters
FORWARDING_PLANE:
    memory-mb   : 4096
    vcpus       : 3
    console_port: 8827
    device-type : virtio

    interfaces  :
      - type      : static
        ipaddr    : 10.85.4.108
        macaddr   : "02:04:17:02:01:02"
        #macaddr   : "0A:00:DD:C0:DE:10"

---
#Interfaces
JUNOS_DEVICES:
    - interface           : ge-0/0/0
      mac-address         : "02:04:17:02:02:01"
      description         : "ge-0/0/0 connects to eth6"

    - interface           : ge-0/0/1
      mac-address         : "02:04:17:02:02:02"
      description         : "ge-0/0/1 connects to eth7"
```

4.3. run installation script with `--cfg` option

After the two config files are ready, run `vmx.sh` to bring them up

```
./vmx.sh -lvf --install --cfg config/vmx.conf.virtio.1
./vmx.sh -lvf --install --cfg config/vmx.conf.virtio.2
```

4.4. verify the 2 running VMX

list the 2 running VMX instances:

vmx1 and vmx2, each contains a vcp and a vfp VM.

```
ping@trinity:/virtualization/images/vmx_20151102.0$ sudo virsh list
 Id    Name                           State
----------------------------------------------------
 2     vcp-vmx1                       running
 3     vfp-vmx1                       running
 5     vcp-vmx2                       running
 6     vfp-vmx2                       running
```

4.4.1. ifconfig (virtio)

Here is the dump of all interfaces after two VMXs are up and running:

```
ping@trinity:/virtualization/images/vmx_20151102.0$ ifconfig -a
br-ext    Link encap:Ethernet  HWaddr 38:ea:a7:37:7c:54               \
          inet addr:10.85.4.17  Bcast:10.85.4.127  Mask:255.255.255.128\
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1            |
          RX packets:26432 errors:0 dropped:0 overruns:0 frame:0        |
          TX packets:10158 errors:0 dropped:0 overruns:0 carrier:0      |
          collisions:0 txqueuelen:0                                     |
          RX bytes:2025742 (2.0 MB)  TX bytes:1184591 (1.1 MB)          |
                                                                        |
br-ext-nic Link encap:Ethernet  HWaddr 52:54:00:9f:a0:77                |
          BROADCAST MULTICAST  MTU:1500  Metric:1                       |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0            |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0          |
          collisions:0 txqueuelen:500                                   |
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                        |
                                                                        |①
vcp_ext-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:01:01              |
          inet6 addr: fe80::fc04:17ff:fe01:101/64 Scope:Link            |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1            |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0            |
          TX packets:19408 errors:0 dropped:0 overruns:0 carrier:0      |
          collisions:0 txqueuelen:500                                   |
          RX bytes:0 (0.0 B)  TX bytes:1826586 (1.8 MB)                 |
                                                                        |
vcp_ext-vmx2 Link encap:Ethernet  HWaddr fe:04:17:02:01:01              |
          inet6 addr: fe80::fc04:17ff:fe02:101/64 Scope:Link            |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1            |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0            |
          TX packets:19248 errors:0 dropped:0 overruns:0 carrier:0      |
          collisions:0 txqueuelen:500                                   |
          RX bytes:0 (0.0 B)  TX bytes:1812426 (1.8 MB)                 |
                                                                        |
vfp_ext-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:01:02              |
          inet6 addr: fe80::fc04:17ff:fe01:102/64 Scope:Link            |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1            |
          RX packets:18 errors:0 dropped:0 overruns:0 frame:0           |
          TX packets:19409 errors:0 dropped:0 overruns:0 carrier:0      |
```

```
          collisions:0 txqueuelen:500                                    |
          RX bytes:2840 (2.8 KB)  TX bytes:1827538 (1.8 MB)              |
                                                                         |
vfp_ext-vmx2 Link encap:Ethernet  HWaddr fe:04:17:02:01:02               |
          inet6 addr: fe80::fc04:17ff:fe02:102/64 Scope:Link             |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1             |
          RX packets:118 errors:0 dropped:0 overruns:0 frame:0           |
          TX packets:19129 errors:0 dropped:0 overruns:0 carrier:0       |
          collisions:0 txqueuelen:500                                     /
          RX bytes:38268 (38.2 KB)  TX bytes:1774106 (1.7 MB)            /

br-int-vmx1 Link encap:Ethernet  HWaddr 52:54:00:56:2f:2b               \
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1             \
          RX packets:12823 errors:0 dropped:12587 overruns:0 frame:0     |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0           |
          collisions:0 txqueuelen:0                                      |
          RX bytes:949948 (949.9 KB)  TX bytes:0 (0.0 B)                 |
                                                                         |
br-int-vmx1-nic Link encap:Ethernet  HWaddr 52:54:00:56:2f:2b            |
          BROADCAST MULTICAST  MTU:1500  Metric:1                        |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0             |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0           |
          collisions:0 txqueuelen:500                                    |
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                         |
                                                                         | ②
vcp_int-vmx1 Link encap:Ethernet  HWaddr fe:54:00:d6:6e:85               |
          inet6 addr: fe80::fc54:ff:fed6:6e85/64 Scope:Link              |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1             |
          RX packets:109078 errors:0 dropped:0 overruns:0 frame:0        |
          TX packets:112829 errors:0 dropped:0 overruns:0 carrier:0      |
          collisions:0 txqueuelen:500                                    |
          RX bytes:6930175 (6.9 MB)  TX bytes:10139042 (10.1 MB)         |
                                                                         |
vfp_int-vmx1 Link encap:Ethernet  HWaddr fe:54:00:f5:4e:ee               |
          inet6 addr: fe80::fc54:ff:fef5:4eee/64 Scope:Link              |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1             |
          RX packets:108951 errors:0 dropped:0 overruns:0 frame:0        |
          TX packets:112338 errors:0 dropped:0 overruns:0 carrier:0      |
          collisions:0 txqueuelen:500                                     /
          RX bytes:9928502 (9.9 MB)  TX bytes:7099927 (7.0 MB)           /

br-int-vmx2 Link encap:Ethernet  HWaddr 52:54:00:14:94:e5               \
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1             \
          RX packets:12638 errors:0 dropped:12463 overruns:0 frame:0     |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0           |
          collisions:0 txqueuelen:0                                      |
          RX bytes:938940 (938.9 KB)  TX bytes:0 (0.0 B)                 |
                                                                         |
br-int-vmx2-nic Link encap:Ethernet  HWaddr 52:54:00:14:94:e5            |
          BROADCAST MULTICAST  MTU:1500  Metric:1                        |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0             |
```

```
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                    ③

vcp_int-vmx2 Link encap:Ethernet  HWaddr fe:54:00:46:01:02
          inet6 addr: fe80::fc54:ff:fe46:102/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:107951 errors:0 dropped:0 overruns:0 frame:0
          TX packets:111724 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:6866493 (6.8 MB)  TX bytes:10053803 (10.0 MB)

vfp_int-vmx2 Link encap:Ethernet  HWaddr fe:54:00:b9:67:1d
          inet6 addr: fe80::fc54:ff:feb9:671d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:107878 errors:0 dropped:0 overruns:0 frame:0
          TX packets:111185 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:9845011 (9.8 MB)  TX bytes:7034893 (7.0 MB)

em1       Link encap:Ethernet  HWaddr 38:ea:a7:37:7c:54
          inet6 addr: fe80::3aea:a7ff:fe37:7c54/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1642746 errors:0 dropped:606 overruns:0 frame:0
          TX packets:15147394 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:139306967 (139.3 MB)  TX bytes:21847662292 (21.8 GB)

em2       Link encap:Ethernet  HWaddr 38:ea:a7:37:7c:55
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

em9       Link encap:Ethernet  HWaddr 38:ea:a7:37:7b:d0
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

em10      Link encap:Ethernet  HWaddr 38:ea:a7:37:7b:d1
          BROADCAST MULTICAST  MTU:1500  Metric:1                  ④
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

p2p1      Link encap:Ethernet  HWaddr 38:ea:a7:17:65:a0
          inet6 addr: fe80::3aea:a7ff:fe17:65a0/64 Scope:Link
```

```
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1          |
          RX packets:5 errors:0 dropped:0 overruns:0 frame:0          |
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0        |
          collisions:0 txqueuelen:1000                               |
          RX bytes:300 (300.0 B)  TX bytes:648 (648.0 B)             |
                                                                     |
p2p2      Link encap:Ethernet  HWaddr 38:ea:a7:17:65:a1              |
          BROADCAST MULTICAST  MTU:1500  Metric:1                    |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0         |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0       |
          collisions:0 txqueuelen:1000                               |
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                     |
                                                                     |
p3p1      Link encap:Ethernet  HWaddr 38:ea:a7:17:65:84              |
          BROADCAST MULTICAST  MTU:1500  Metric:1                    |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0         |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0       |
          collisions:0 txqueuelen:1000                               |
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                     |
                                                                     |
p3p2      Link encap:Ethernet  HWaddr 38:ea:a7:17:65:85              |
          BROADCAST MULTICAST  MTU:1500  Metric:1                    |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0         |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0       |
          collisions:0 txqueuelen:1000                              /
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                   /

ge-0.0.0-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:02:01    \
          inet6 addr: fe80::fc04:17ff:fe01:201/64 Scope:Link    \
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1     |
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0     |
          TX packets:3263 errors:0 dropped:0 overruns:0 carrier:0|
          collisions:0 txqueuelen:500                            |
          RX bytes:280 (280.0 B)  TX bytes:169990 (169.9 KB)     |
                                                                 |
ge-0.0.0-vmx2 Link encap:Ethernet  HWaddr fe:04:17:02:02:01      |
          inet6 addr: fe80::fc04:17ff:fe02:201/64 Scope:Link     |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1     |
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0     |
          TX packets:3238 errors:0 dropped:0 overruns:0 carrier:0\
          collisions:0 txqueuelen:500                             X ⑤
          RX bytes:238 (238.0 B)  TX bytes:168680 (168.6 KB)     /
                                                                 |
ge-0.0.1-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:02:02      |
          inet6 addr: fe80::fc04:17ff:fe01:202/64 Scope:Link     |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1     |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0     |
          TX packets:3262 errors:0 dropped:0 overruns:0 carrier:0|
          collisions:0 txqueuelen:500                            |
          RX bytes:0 (0.0 B)  TX bytes:169836 (169.8 KB)         |
                                                                 |
```

```
ge-0.0.1-vmx2 Link encap:Ethernet  HWaddr fe:04:17:02:02:02      |
          inet6 addr: fe80::fc04:17ff:fe02:202/64 Scope:Link     |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1     |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0     |
          TX packets:3236 errors:0 dropped:0 overruns:0 carrier:0|
          collisions:0 txqueuelen:500                            |
          RX bytes:0 (0.0 B)  TX bytes:168484 (168.4 KB)        /

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:424272 errors:0 dropped:0 overruns:0 frame:0
          TX packets:424272 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:22393676 (22.3 MB)  TX bytes:22393676 (22.3 MB)

virbr0    Link encap:Ethernet  HWaddr fe:04:17:01:02:01
          inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:30 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2751 (2.7 KB)  TX bytes:812 (812.0 B)
```

① external bridge

② internal bridge for vmx1

③ internal bridge for vmx2

④ physical interfaces

⑤ virtio tap interfaces, peer interfaces [1: interface showing in host, and peering with guest VM interface, e.g. the `vcp_ext-vmx1` interface is "peer interface" of fxp0 in VMX] of JUNOS ge-0/0/x interfaces in VFP VM

4.4.2. virtio bridging: linux bridge

virtio does not define anything special for interface bridging. Any open techniques (linux brige, OVS, etc) can be used for this purpose.

A separate config file `vmx-junosdev.conf` is used to automate virtio interface bridging using native linux bridge utility.

in this example, I'll define a new bridge named `bridge1`, and put following interfaces into that same bridge:

- ge-0/0/0 in vmx1
- ge-0/0/0 in vmx2
- p2p1

The `vmx-junosdev.conf` file:

```
ping@trinity:/virtualization/images/vmx_20151102.0$ cat config/vmx-junosdev.conf.1
#################################################################
#
#  vmx-junos-dev.conf
#  - Config file for junos device bindings.
#  - Uses YAML syntax.
#  - Leave a space after ":" to specify the parameter value.
#  - For physical NIC, set the 'type' as 'host_dev'
#  - For junos devices, set the 'type' as 'junos_dev' and
#    set the mandatory parameter 'vm-name' to the name of
#    the vPFE where the device exists
#  - For bridge devices, set the 'type' as 'bridge_dev'
#
#################################################################
interfaces :

     - link_name  : vmx_bridge1      \
       mtu        : 1500             |
       endpoint_1 :                  |
         - type        : junos_dev   \
           vm_name     : vmx1         X  ①
           dev_name    : ge-0/0/0    /
       endpoint_2 :                  |
         - type        : bridge_dev  |
           dev_name    : bridge1    /

     - link_name  : vmx_bridge1      \
       mtu        : 1500             |
       endpoint_1 :                  |
         - type        : junos_dev   \
           vm_name     : vmx2         X  ②
           dev_name    : ge-0/0/0    /
       endpoint_2 :                  |
         - type        : bridge_dev  |
           dev_name    : bridge1    /

     - link_name  : vmx_bridge1      \
       mtu        : 1500             |
       endpoint_1 :                  |
         - type        : host_dev    \
           dev_name    : p2p1         X  ③
       endpoint_2 :                  /
         - type        : bridge_dev  |
           dev_name    : bridge1    /
```

① put junos interface (type `junos_dev`) `ge-0/0/0` of first VMX instance `vmx1` ,into linux bridge `bridge1`

② put junos interface (type `junos_dev`) `ge-0/0/0` of second VMX instance `vmx2` ,into same linux bridge `bridge1`

③ put host server interface (type `host_dev`) `p2p1` into same linux bridge `bridge1`

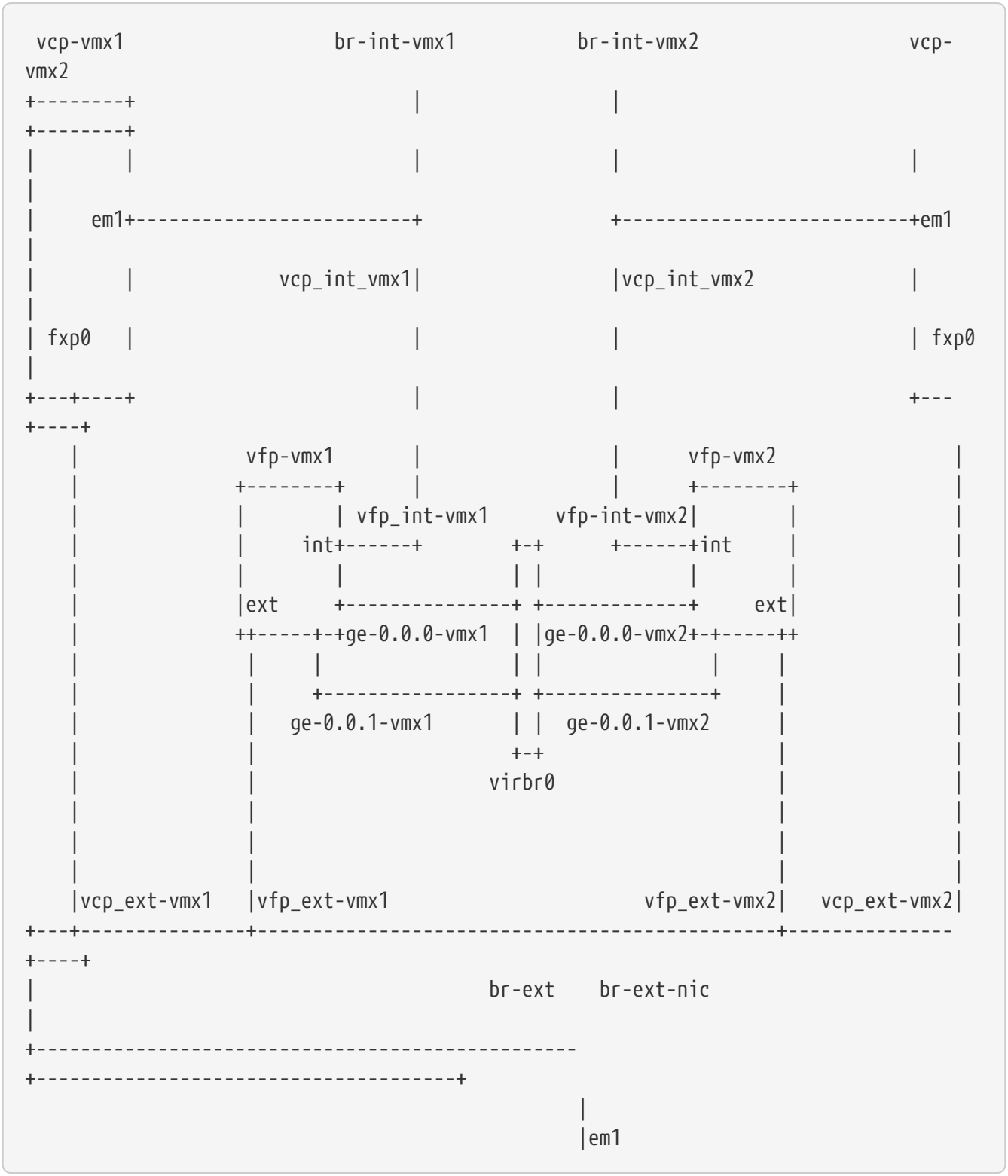
bridging before binding:

before `binding` interfaces to `bridge1`, all four virtio tap interfaces were under `virbr0` interface, which represents the default libvirt network.

```
ping@trinity:/virtualization/images/vmx_20151102.0$ brctl show

bridge name     bridge id               STP enabled     interfaces

br-ext          8000.38eaa7377c54       yes             br-ext-nic
                                                        em1
                                                        vcp_ext-vmx1
                                                        vcp_ext-vmx2
                                                        vfp_ext-vmx1
                                                        vfp_ext-vmx2

br-int-vmx1             8000.525400562f2b       yes     br-int-vmx1-nic
                                                        vcp_int-vmx1
                                                        vfp_int-vmx1

br-int-vmx2             8000.5254001494e5       yes     br-int-vmx2-nic
                                                        vcp_int-vmx2
                                                        vfp_int-vmx2

virbr0          8000.fe0417010201       yes             ge-0.0.0-vmx1
                                                        ge-0.0.0-vmx2
                                                        ge-0.0.1-vmx1
                                                        ge-0.0.1-vmx2
```

the bridges diagram

The current bridging structure in this server is illustrated below:



Now we run the `vmx.sh` script with `--bind-dev` option and `--cfg` option:

```
ping@trinity:/virtualization/images/vmx_20151102.0$ sudo ./vmx.sh --bind-dev --cfg
config/vmx-junosdev.conf.1
Checking package ethtool.........................[OK]
Bind Bridge port bridge1(ge-0.0.0-vmx1)...........[OK]
Bind Bridge port bridge1(ge-0.0.0-vmx2)...........[OK]
Bind Bridge port bridge1(p2p1)....................[OK]
```

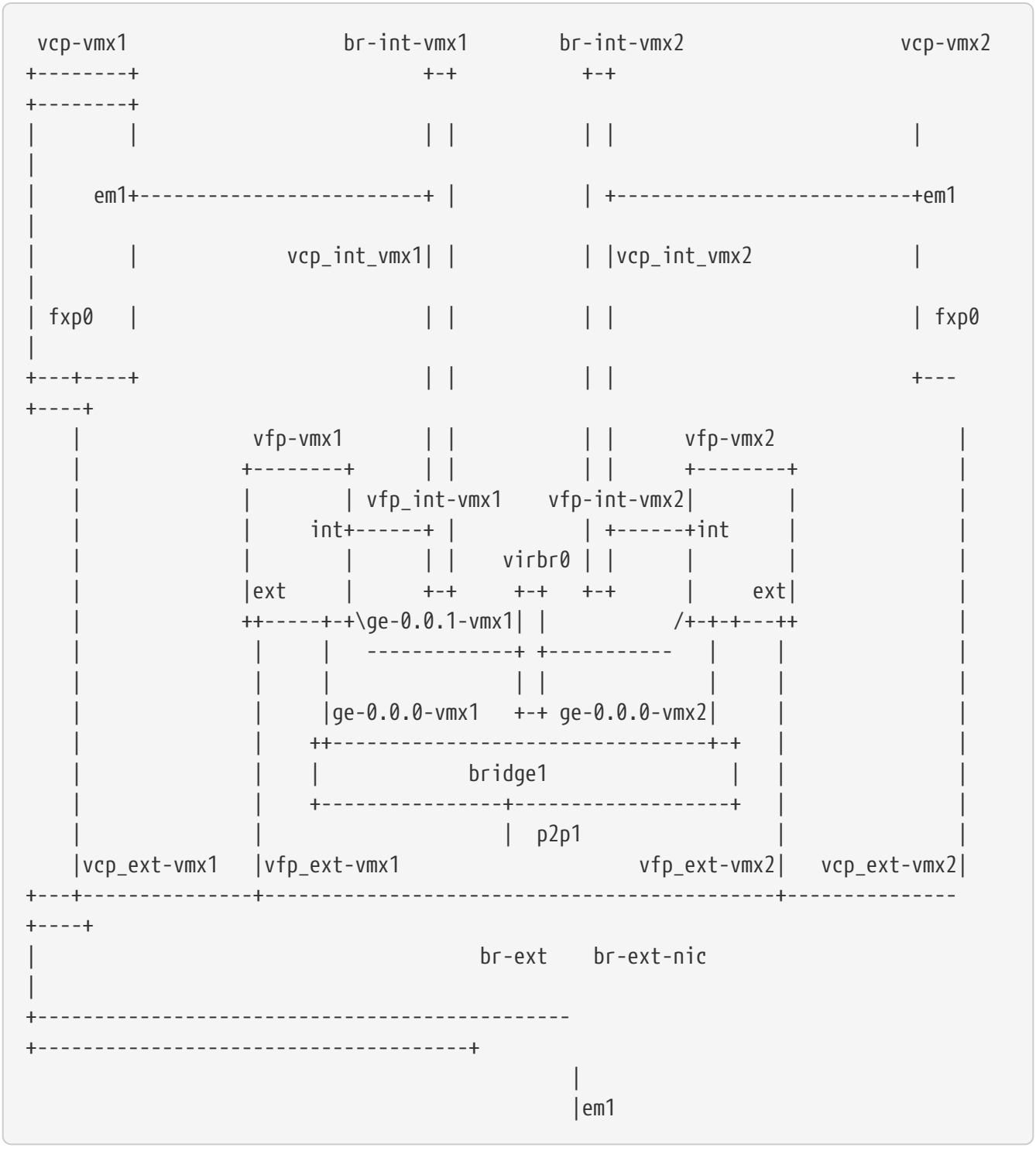
bridging after binding:

After running the `vmx.sh` script, the peer interface [peer interface] of JUNOS interface `ge-0/0/0` from each VMX VM: `ge-0.0.0-vmx1` and `ge-0.0.0-vmx2`, along with the physical port `p2p1`, now is bound to my new bridge `bridge1`.

```
ping@trinity:/virtualization/images/vmx_20151102.0$ brctl show
bridge name     bridge id               STP enabled     interfaces
br-ext          8000.38eaa7377c54       yes             br-ext-nic
                                                        em1
                                                        vcp_ext-vmx1
                                                        vcp_ext-vmx2
                                                        vfp_ext-vmx1
                                                        vfp_ext-vmx2

br-int-vmx1             8000.525400562f2b       yes     br-int-vmx1-nic
                                                        vcp_int-vmx1
                                                        vfp_int-vmx1

br-int-vmx2             8000.5254001494e5       yes     br-int-vmx2-nic
                                                        vcp_int-vmx2
                                                        vfp_int-vmx2

bridge1         8000.38eaa71765a0       no              ge-0.0.0-vmx1
                                                        ge-0.0.0-vmx2
                                                        p2p1

virbr0          8000.fe0417010202       yes             ge-0.0.1-vmx1
                                                        ge-0.0.1-vmx2
```

the bridges diagram

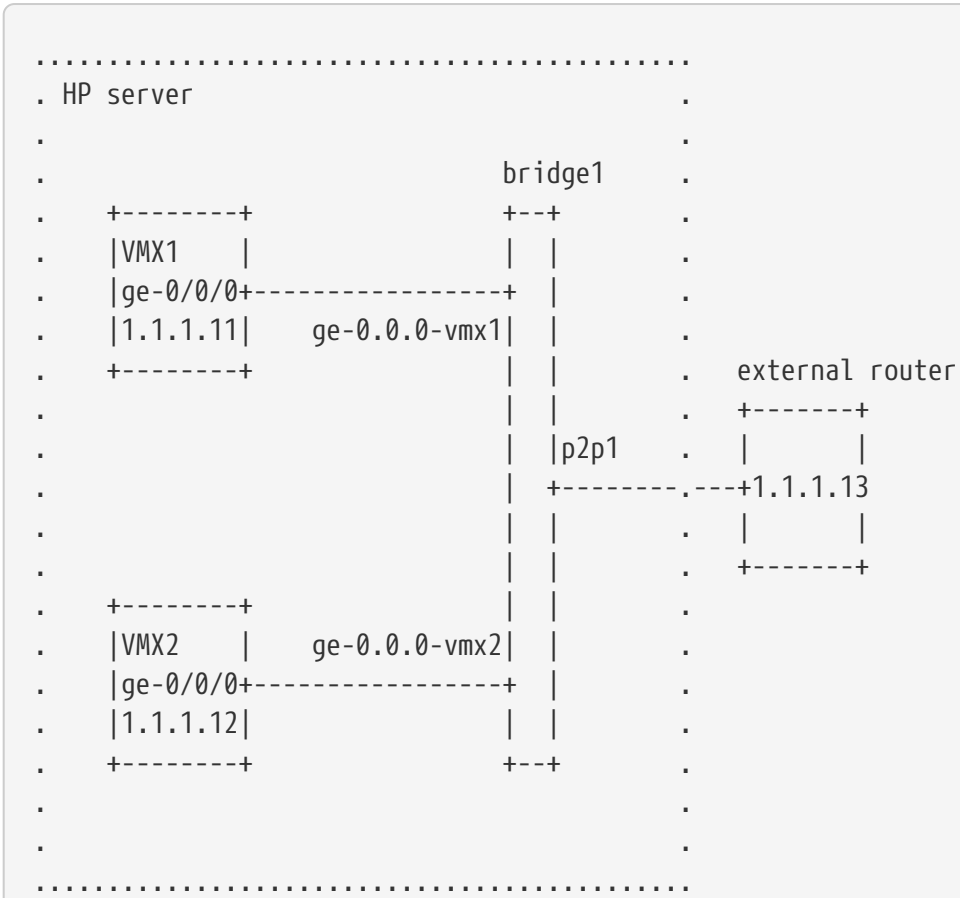
Now the briding structure is changed, as can be illustrated from below diagram, the packet coming from external device, will be bridged to the tap interface `ge-0.0.0-vmx1` and `ge-0.0.0-vmx2` via bridge `bridge1`, and then goes to the corresponding `ge-0/0/0` JUNOS interface in the VMX VM.



4.4.3. routing test

A quick way to verify the 2 VMX instances and bridging structure between them is to run OSPF protocol and ping reachability test between them.

For that purpose I enabled IP/OSPF in the external layer 3 switch attached to the p2p1 interface of the server where VMX instances were built.



In my setup , both ospf adjacency and ping looks good from first VMX instance vmx1. This indicates both instances are running well.

```
[edit]
root@vmx1# run ping 1.1.1.12
PING 1.1.1.12 (1.1.1.12): 56 data bytes
64 bytes from 1.1.1.12: icmp_seq=0 ttl=64 time=3.081 ms
^C
--- 1.1.1.12 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 3.081/3.081/3.081/0.000 ms

[edit]
root@vmx1# run ping 1.1.1.13
PING 1.1.1.13 (1.1.1.13): 56 data bytes
64 bytes from 1.1.1.13: icmp_seq=0 ttl=64 time=16.046 ms

root@vmx1# run show ospf neighbor
Address          Interface              State     ID               Pri  Dead
1.1.1.13         ge-0/0/0.0             Full      30.0.0.1         128    32
1.1.1.12         ge-0/0/0.0             Full      1.1.1.12         128    32

[edit]
root@vmx1# run show route

inet.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

1.1.1.0/24         *[Direct/0] 00:55:52
                    > via ge-0/0/0.0
1.1.1.11/32        *[Local/0] 00:55:52
                      Local via ge-0/0/0.0
224.0.0.5/32       *[OSPF/10] 00:02:50, metric 1
```

Chapter 5. install VMX manually

The installation script is handy to set up VMX, because it helps to automate most of the configuration tasks, which otherwise all need to be done manually.

On the other hand, the script does nothing but to prepare the work environment for the VMX and run some commands to bring up the two VM images. Therefore, having the script does not stop us from doing all of the installation work manually. One advantage of doing so is that if for any reason the script bails out (e.g sometime the server does not meet all of the presumptions or criterias the script requires in order to run smoothly), we can still fine tune the system manually and proceed the installation task. But in order to do that, we need to know what exactly the script does so we can at least use the step as a good and tested-and-working reference when something went wrong in the middle.

Here is the steps the installation script go through:

1. prepare the environment before start:
    - copy all images into a seperate working folder
    - check if all required software packages are installed
2. if not done yet, re-compile the ixgbe driver kernel module from source code for SR-IOV
3. check if there are currently running VMX VMs and bridges with the same name, if yes,destroy and undefine any VMs and internal bridges with same name
4. remove all VFs if existed, by reloading ixgbe kernel module without `max_vfs` option
5. configure "hugepage" and libvirt security and prepare for the VT-d/PCI passthough ("PCI stub")
6. configure VF feature by reloading ixgbe kernel module with `max_vfs` option
7. restart libvirt service daemon to make sure libvirt security is successfully enabled
8. run a python script to parse the YAML config file (`vmx.conf`) and generate all necessary XML files which will be used later by libvirt (virsh) to bring up the VMs. The python script also generate some shell scripts (group of some commands), which can be used to tune the VM properties (vCPU pinning, VF MAC, etc) later.
9. for SRIOV-based VMX only, run one of the generated shell script `vfconfig-generated.sh`, to configure the property of physical port and VFs. the script also use VT-d/PCI passthough feature to detach VFs from the host.
10. define external / internal bridge
11. define and bring up VMX VMs
12. perform vCPU pinning

The running log of the installation script, especially with `-lvf` debug option, reveals commands and steps to prepare and setup VMX.

5.1. script generated XML files

the installation script will generate some XML and shell scripts:

- all necessary libvirt XML files

  this will be used by the virsh tool to bring up the VMX.

```
ping@trinity:/virtualization/images/vmx_20151102.0/build/vmx1/xml$ ls -l | grep xml
-rw-r--r-- 1 root root  383 Nov 24 22:23 br-ext-generated.xml
-rw-r--r-- 1 root root   99 Nov 24 22:23 br-int-generated.xml
-rw-r--r-- 1 root root 2749 Nov 24 22:23 vPFE-generated.xml
-rw-r--r-- 1 root root 3560 Nov 24 22:23 vRE-generated.xml
```

- shell scripts

  these will be executed by the installation script to configure VF properties and to configure vCPU pinning

```
ping@trinity:/virtualization/images/vmx_20151102.0/build/vmx1/xml$ ls -l | grep sh
-rw-r--r-- 1 root root  226 Nov 24 22:23 cpu_affinitize.sh
-rw-r--r-- 1 root root  712 Nov 24 22:23 vfconfig-generated.sh
```

prepare the XML files for libvirt/virsh

The XML files play an crucial role when bringing up VMs using libvirt/virsh tool - the libvirt tool relies on XML to store structured data. Numerous API functions of libvirt (and their implementation in virsh) take XML documents as their arguments. XML documents are passed to the XML parser, that detects syntax errors and then are processed internaly.

We need to have the XML files ready for use before start to install VMX manually.

1. create all the necessary XML files

   XML files can be generated by any of the below methods:

   - modify existing files, which could be generated by VMX installation script in other setup (recommended)
   - creat them from scratch (not recommeneded)

2. update all parameters in the XML files
   - domain/bridge names
   - image path
   - MAC of ext mgmt (vcp_ext-vmx) interface in external bridge
   - console/serial port
   - for VFP VM: hostdev (SR-IOV VF) function# and MAC#

- external bridge xml: external MAC and IP of mgmt i/f (vcp_ext-vmx)

5.2. manual installation steps

1. enable iommu/VT-d (SR-IOV only)

   enabling IOMMU/VT-d allows the physical PCI devices to be able to be assigned to the VM directly.

   skip if this was done once

2. recompile and reload ixgbe kernel driver module (SR-IOV only)

   Juniper modified IXGBE kernel driver is required for VMX. the driver needs to be re-compiled from source code that comes with the installation package.

   skip if this was done once

   to verify if the ixgbe version is right:

   ```
   ping@ubuntu1:~$ modinfo ixgbe | grep version
   version:        3.19.1
   srcversion:     B97B1E7CF79A25F5E4D7B96
   vermagic:       3.13.0-32-generic SMP mod_unload modversions
   ```

   - compile and install ixgbe driver from source code

   ```
   cd /virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src
   rm -f ixgbe.ko
   make install
   sleep 5
   cmp ixgbe.ko /lib/modules/3.13.0-32-
   generic/kernel/drivers/net/ethernet/intel/ixgbe/ixgbe.ko
   ```

   - reload ixgbe driver

   ```
   rmmod ixgbevf
   rmmod ixgbe
   modprobe ixgbe
   ```

3. setup hugepage

   skip if this was done once

```
sudo -i
echo 32768 > /proc/sys/vm/nr_overcommit_hugepages
exit

mkdir /HugePage_vPFE
sudo mount -t hugetlbfs hugetlbfs /HugePage_vPFE
sudo mount | grep "HugePage_vPFE"

sudo service libvirt-bin restart

cat /etc/apparmor.d/abstractions/libvirt-qemu  | grep HugePage_vPFE
sudo -i
echo "owner \"/HugePage_vPFE/libvirt/qemu/**\" rw," >>
/etc/apparmor.d/abstractions/libvirt-qemu
exit
```

- VM won't start properly without hugetlbfs mounted beforehand.

```
ping@matrix:~$ ls /HugePage_vPFE/
ping@matrix:~$ mount | grep "HugePage_vPFE"
ping@matrix:~$
ping@matrix:~$ sudo virsh start AVPN-vfp
error: Failed to start domain AVPN-vfp
error: internal error: hugetlbfs filesystem is not mounted or
disabled by administrator config
```

- libvirt needs to be restarted after hugetlbfs got mounted:

```
ping@matrix:~$ sudo mount -t hugetlbfs hugetlbfs /HugePage_vPFE
ping@matrix:~$ sudo virsh start AVPN-vfp
error: Failed to start domain AVPN-vfp
error: internal error: hugetlbfs filesystem is not mounted or
disabled by administrator config

ping@matrix:~$ ls /HugePage_vPFE/

ping@matrix:~$ sudo service libvirt-bin restart
libvirt-bin stop/waiting
libvirt-bin start/running, process 46220

ping@matrix:~$ ls /HugePage_vPFE/
libvirt
ping@matrix:~$ sudo virsh start AVPN-vfp
Domain AVPN-vfp started
```

4. prepare for pci-stub (SR-IOV only)

skip if this was done once

```
sudo modprobe pci-stub

sudo -i
echo "8086 10ed" > /sys/bus/pci/drivers/pci-stub/new_id
> /sys/bus/pci/drivers/pci-stub/bind
exit
```

5. reload ixgbe kernel driver and configure SR-IOV VF (SR-IOV only)

```
sudo rmmod ixgbevf; sudo rmmod ixgbe; \
sudo modprobe ixgbe max_vfs=2,2,2,2,2,2,2,2; \
sudo modprobe tun ; \                                    ①
sleep 5; sudo brctl addif br-ext em9
```

① tun/tap interface: TUN (namely network TUNnel) simulates a network layer device and it operates with layer 3 packets like IP packets. TAP (namely network tap) simulates a link layer device and it operates with layer 2 packets like Ethernet frames. TUN is used with routing, while TAP is used for creating a network bridge, see wiki. In VMX "tap" interface and bridges will be used and that is why this kernel module is required. this is not directly related to SR-IOV.

these VF configs, and also a lot of the other configuration here won't be persistent across system reboot.

6. configure interface and VF properties (SR-IOV only)

follow the MAC assignment rule to assign MAC for the 2 VMX instance

MAC address assignment

```
02:04:17:01:01:01 vcp-vmx1 fxp0
02:04:17:01:01:02 vfp-vmx1 eth0
02:04:17:01:02:01 vmx1 ge-0/0/0 p3p1 vf0
02:04:17:01:02:02 vmx1 ge-0/0/1 p2p1 vf0

02:04:17:02:01:01 vcp-vmx2 fxp0
02:04:17:02:01:02 vfp-vmx2 eth0
02:04:17:02:02:01 vmx2 ge-0/0/0 p3p1 vf1
02:04:17:02:02:02 vmx2 ge-0/0/1 p2p1 vf1
```

```
export XML_FOLDER="/virtualization/vmx1/xml"
sudo sh $XML_FOLDER/vfconfig-generated.sh
export XML_FOLDER="/virtualization/vmx2/xml"
sudo sh $XML_FOLDER/vfconfig-generated.sh
```

or

```
#VMX1
sudo ifconfig p3p1 up promisc allmulti mtu 2000
sudo ifconfig p2p1 up promisc allmulti mtu 2000
sudo ip link set p3p1 vf 0 mac 02:04:17:01:02:01
sudo ip link set p2p1 vf 0 mac 02:04:17:01:02:02
sudo ip link set p3p1 vf 0 rate 10000
sudo ip link set p2p1 vf 0 rate 10000
sudo ip link set p3p1 vf 0 spoofchk off
sudo ip link set p2p1 vf 0 spoofchk off

#VMX2
sudo ifconfig p3p1 up promisc allmulti mtu 2000
sudo ifconfig p2p1 up promisc allmulti mtu 2000
sudo ip link set p3p1 vf 1 mac 02:04:17:02:02:01
sudo ip link set p2p1 vf 1 mac 02:04:17:02:02:02
sudo ip link set p3p1 vf 1 rate 10000
sudo ip link set p2p1 vf 1 rate 10000
sudo ip link set p3p1 vf 1 spoofchk off
sudo ip link set p2p1 vf 1 spoofchk off
```

7. pci-stub hide (SR-IOV only)

   unbind the VF from kernel driver (host) and bind it to pci-stub module, effectively hide this VF from host kernel and the VF will later be assigned directly to the VM.

   skip if this was done once

```
sudo -i
#p3p1 vf0
echo 0000:23:10.0 > /sys/bus/pci/devices/0000:23:10.0/driver/unbind
echo 0000:23:10.0 >> /sys/bus/pci/drivers/pci-stub/bind
#p3p1 vf1
echo 0000:23:10.2 > /sys/bus/pci/devices/0000:23:10.2/driver/unbind
echo 0000:23:10.2 >> /sys/bus/pci/drivers/pci-stub/bind

#p2p1 vf0
echo 0000:06:10.0 > /sys/bus/pci/devices/0000:06:10.0/driver/unbind
echo 0000:06:10.0 >> /sys/bus/pci/drivers/pci-stub/bind
#p2p1 vf1
echo 0000:06:10.2 > /sys/bus/pci/devices/0000:06:10.2/driver/unbind
echo 0000:06:10.2 >> /sys/bus/pci/drivers/pci-stub/bind
exit
```

8. config VN & bridges

```
#vmx1:
export XML_FOLDER="/virtualization/vmx1"
sudo virsh net-define $XML_FOLDER/br-ext-generated.xml

sudo ifconfig em1 0; \
sudo virsh net-start br-ext; \
sudo brctl addif br-ext em1; \
sudo route add default gw 10.85.4.1
sudo ifconfig br-ext hw ether 38:ea:a7:37:7c:54;

sudo virsh net-define $XML_FOLDER/br-int-generated.xml
sudo virsh net-start br-int-vmx1

#vmx2:
export XML_FOLDER="/virtualization/vmx2"

sudo virsh net-define $XML_FOLDER/br-int-generated.xml
sudo virsh net-start br-int-vmx2
```

9. define and start VMs:

```
sudo virsh net-list | grep default

#vmx1
sudo virsh define /virtualization/vmx1/vRE-generated1.xml
sudo virsh define /virtualization/vmx1/vPFE-generated1.xml
sudo virsh start vcp-vmx1
sudo virsh start vfp-vmx1

#vmx2
sudo virsh define /virtualization/vmx2/vRE-generated2.xml
sudo virsh define /virtualization/vmx2/vPFE-generated2.xml
sudo virsh start vcp-vmx2
sudo virsh start vfp-vmx2
```

10. vcpupin:

```
export XML_FOLDER="/virtualization/vmx1"
sudo sh $XML_FOLDER/cpu_affinitize.sh
export XML_FOLDER="/virtualization/vmx2"
sudo sh $XML_FOLDER/cpu_affinitize.sh
```

or

```
#vmx1:
sudo virsh emulatorpin vcp-vmx1 0
sudo virsh emulatorpin vfp-vmx1 0

sudo virsh vcpupin vcp-vmx1 0 7

sudo virsh vcpupin vfp-vmx1 0 0
sudo virsh vcpupin vfp-vmx1 1 1
sudo virsh vcpupin vfp-vmx1 2 2
sudo virsh vcpupin vfp-vmx1 3 3

#vmx2:
sudo virsh emulatorpin vcp-vmx2 0
sudo virsh emulatorpin vfp-vmx2 0

sudo virsh vcpupin vcp-vmx2 0 15

sudo virsh vcpupin vfp-vmx2 0 8
sudo virsh vcpupin vfp-vmx2 1 9
sudo virsh vcpupin vfp-vmx2 2 10
sudo virsh vcpupin vfp-vmx2 3 11
```

multiple VMX instances

using the above files generated from the installation script, it's easy to bring up multiple instances of VMX manually.

5.3. verify installations

show running VMX instances:

```
ping@trinity:/virtualization/vmx1$ sudo virsh list --all
 Id    Name                           State
----------------------------------------------------
 4     vcp-vmx1                       running
 6     vfp-vmx1                       running
 7     vcp-vmx2                       running
 8     vfp-vmx2                       running
```

identify console ports of VCP and VFP VMs for both instance:

```
ping@trinity:/virtualization/vmx1$ netstat -na | grep ":8[12][67] "
tcp        0      0 127.0.0.1:8826          0.0.0.0:*               LISTEN
#<------
tcp        0      0 127.0.0.1:8827          0.0.0.0:*               LISTEN
#<------
tcp        0      0 127.0.0.1:8816          0.0.0.0:*               LISTEN
#<------
tcp        0      0 127.0.0.1:8817          0.0.0.0:*               LISTEN
#<------
tcp        0      0 127.0.0.1:8816          127.0.0.1:42765         ESTABLISHED
tcp        0      0 127.0.0.1:41726         127.0.0.1:8826          ESTABLISHED
tcp        0      0 127.0.0.1:42765         127.0.0.1:8816          ESTABLISHED
tcp        0      0 127.0.0.1:8826          127.0.0.1:41726         ESTABLISHED
```

the port number that qemu is listening can be configured to any number, as long as the port is unique in the system wide and not in use by other applications. in the above example I use 88x6 for VCP console port, and 88x7 for VFP console port, where x is instance number.

Another approach, is to use the same port number for all instances, but changing the local IP to 127.0.0.x where x can be instance number - just make sure the local "socket" (IP + port pair) is unique will be sufficient.

```
ping@ubuntu4.54:~$ sudo netstat -nap | grep qemu
[sudo] password for ping:
tcp        0      0 127.0.0.2:8896          0.0.0.0:*
LISTEN      33284/qemu-system-x     #<------
tcp        0      0 127.0.0.1:8896          0.0.0.0:*
LISTEN      15030/qemu-system-x     #<------
tcp        0      0 127.0.0.2:8897          0.0.0.0:*
LISTEN      33330/qemu-system-x     #<------
tcp        0      0 127.0.0.1:8897          0.0.0.0:*
LISTEN      15075/qemu-system-x     #<------
tcp        0      0 127.0.0.1:5900          0.0.0.0:*
LISTEN      15030/qemu-system-x
tcp        0      0 127.0.0.1:5901          0.0.0.0:*
LISTEN      15075/qemu-system-x
tcp        0      0 127.0.0.2:5902          0.0.0.0:*
LISTEN      33284/qemu-system-x
tcp        0      0 127.0.0.2:5903          0.0.0.0:*
LISTEN      33330/qemu-system-x
unix  2      [ ACC ]     STREAM     LISTENING     62021    15030/qemu-
system-x //lib/libvirt/qemu/MIS-VMX-VCP.monitor
unix  2      [ ACC ]     STREAM     LISTENING     23706    15075/qemu-
system-x //lib/libvirt/qemu/MIS-VMX-VFP.monitor
unix  2      [ ACC ]     STREAM     LISTENING     62277    33284/qemu-
system-x //lib/libvirt/qemu/AVPN-VMX-VCP.monitor
unix  2      [ ACC ]     STREAM     LISTENING     24867    33330/qemu-
system-x //lib/libvirt/qemu/AVPN-VMX-VFP.monitor
unix  3      [ ]         STREAM     CONNECTED     63371    33330/qemu-
system-x //lib/libvirt/qemu/AVPN-VMX-VFP.monitor
unix  3      [ ]         STREAM     CONNECTED     27800    15030/qemu-
system-x //lib/libvirt/qemu/MIS-VMX-VCP.monitor
unix  3      [ ]         STREAM     CONNECTED     63366    33284/qemu-
system-x //lib/libvirt/qemu/AVPN-VMX-VCP.monitor
unix  3      [ ]         STREAM     CONNECTED     59462    15075/qemu-
system-x //lib/libvirt/qemu/MIS-VMX-VFP.monitor
```

login to console of the two VMX instances:

vmx1:

```
ping@trinity:~$ telnet localhost 8816
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Amnesiac (ttyd0)

login: root

--- JUNOS 15.1F-20151104.0 built 2015-11-04 05:39:37 UTC
root@% cli
root >

root> show version
Model: vmx
Junos: 15.1F-20151104.0
JUNOS Base OS boot [15.1F-20151104.0]
JUNOS Base OS Software Suite [15.1F-20151104.0]
...

labroot> show chassis fpc
                     Temp  CPU Utilization (%)   CPU Utilization (%)  Memory
Utilization (%)
Slot State            (C)  Total  Interrupt      1min   5min   15min  DRAM (MB) Heap
Buffer
  0  Online           Testing   3         0        3      3      2      1         6
0

labroot> show chassis fpc pic-status
Slot 0   Online       Virtual FPC
  PIC 0  Online       Virtual
```

vmx2:

```
ping@trinity:~$ telnet localhost 8826
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
......
Amnesiac (ttyd0)
login: lab
root@% cli
root>
```

ping test:

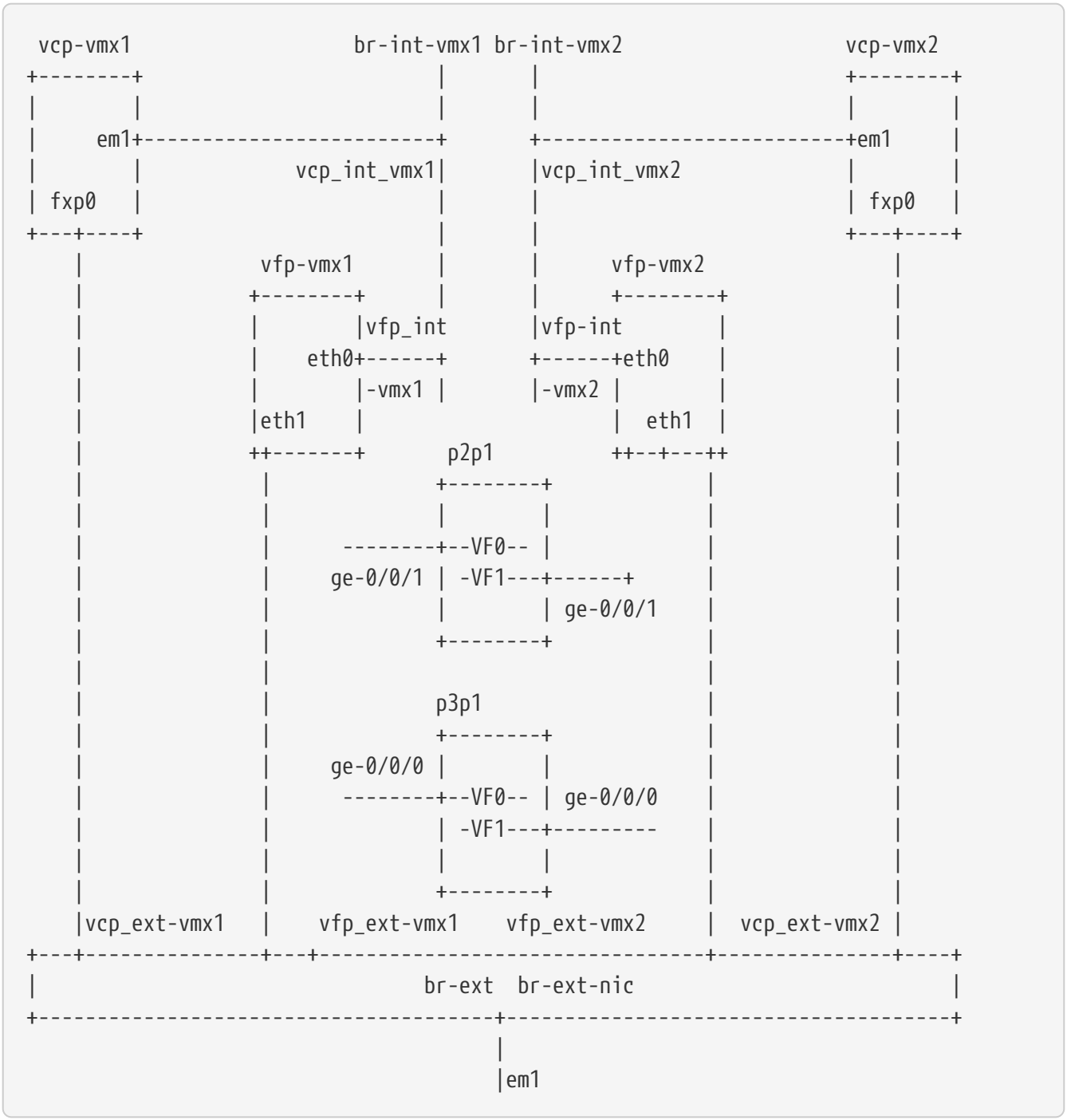
```
labroot@vmx1# run show interfaces ge-0/0/0 terse
Interface               Admin Link Proto    Local                 Remote
ge-0/0/0                up    up
ge-0/0/0.0              up    up   inet     1.1.1.1/24
                                  multiservice

root@vmx2# run show interfaces ge-0/0/0 terse
Interface               Admin Link Proto    Local                 Remote
ge-0/0/0                up    up
ge-0/0/0.0              up    up   inet     1.1.1.2/24
                                  multiservice

[edit]
labroot@vmx1# run ping 1.1.1.2
PING 1.1.1.2 (1.1.1.2): 56 data bytes
64 bytes from 1.1.1.2: icmp_seq=0 ttl=64 time=1.694 ms
64 bytes from 1.1.1.2: icmp_seq=1 ttl=64 time=1.751 ms
64 bytes from 1.1.1.2: icmp_seq=2 ttl=64 time=1.817 ms
^C
--- 1.1.1.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.694/1.754/1.817/0.050 ms

[edit]
labroot@vmx1# run show arp
MAC Address       Address         Name                      Interface
Flags
02:04:17:02:02:01 1.1.1.2         1.1.1.2                   ge-0/0/0.0
none
52:54:00:9c:b3:f2 128.0.0.16      128.0.0.16                em1.0
none
Total entries: 2

[edit]
labroot@vmx1# run ping 10.85.4.107
PING 10.85.4.107 (10.85.4.107): 56 data bytes
64 bytes from 10.85.4.107: icmp_seq=0 ttl=64 time=0.822 ms
64 bytes from 10.85.4.107: icmp_seq=1 ttl=64 time=0.794 ms
^C
--- 10.85.4.107 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.794/0.808/0.822/0.014 ms
```

5.4. internal and external bridges in multiple tenant environment

some highlights of the multiple VMX installation process we demonstrated:

- external management interfaces of all VMs from both instance share same bridge: `br-ext`
- each instance has its own internal bridge `br-int-vmx1` and `br-int-vmx2` for internal management interfaces of VCP/VFP VM
- the physical NIC port p2p1 and p3p1 were each "split" into two virtual port (VF), and each VF were used by a different instance.

These can be illustrated in this diagram:



5.5. manually uninstall VMXs

the steps to "uninstall" VMX are much simpler:

1. "poweroff" and remove VMs:

```
sudo virsh destroy vcp-vmx1
sudo virsh destroy vfp-vmx1
sudo virsh undefine vcp-vmx1
sudo virsh undefine vfp-vmx1
```

```
sudo virsh destroy vcp-vmx2
sudo virsh destroy vfp-vmx2
sudo virsh undefine vcp-vmx2
sudo virsh undefine vfp-vmx2
```

2. disable and delete the VNs and associated bridges:

```
sudo virsh net-destroy br-ext; \
sudo ifconfig em1 10.85.4.17/25; \
sudo route add default gw 10.85.4.1
sudo virsh net-undefine br-ext
```

```
sudo virsh net-destroy br-int-vmx1
sudo virsh net-undefine br-int-vmx1
sudo virsh net-destroy br-int-vmx2
sudo virsh net-undefine br-int-vmx2
```

3. disable VF of SR-IOV:

```
sudo rmmod ixgbevf; \
sudo rmmod ixgbe; \
sleep 5
sudo modprobe ixgbe
```

Chapter 6. upgrading VMX

The procedure of upgrading VMX is conceptually very similiar with the procedure of upgrading physical Junos router - you specify the location of the new image and reboot the box with it.

1. backup VCP/VFP VM configuration XML files:
   a. virsh dumpxml vfp-vmx1 > vfp-vmx1.xml
   b. virsh dumpxml vcp-vmx1 > vcp-vmx1.xml
   c. virsh net-dumpxml br-int-vmx1 > br-int-vmx1.xml
2. Edit the generated vfp-vmx1.xml & vcp-vmx1.xml so they point to desired JUNOS image, PFE image, and HDD file

   VCP

```
<disk device="disk" type="file">
  <driver cache="directsync" name="qemu" type="qcow2" />
  <source file="/path/to/jinstall64-vmx-15.1F-20151104.0-domestic.img" />
#<------
  <target bus="ide" dev="hda" />
  <address bus="0" controller="0" target="0" type="drive" unit="0" />
</disk>

<disk device="disk" type="file">
  <driver cache="directsync" name="qemu" type="qcow2" />
  <source file="/path/to/vmxhdd.img" />     #<------
  <target bus="ide" dev="hdb" />
  <address bus="0" controller="0" target="0" type="drive" unit="1" />
</disk>
```

   VFP

```
<disk device="disk" type="file">
  <driver cache="directsync" name="qemu" type="raw" />
  <source file="/images/vmx_20151102.0/build/vmx1/images/vFPC-20151102.img" />
#<------
  <target bus="ide" dev="hda" />
</disk>
```

3. shutdown VCP/VFP VMs
   a. virsh destroy vfp-vmx1
   b. virsh destroy vcp-vmx1
   c. virsh undefine vfp-vmx1
   d. virsh undefine vcp-vmx1
   e. virsh net-undefine br-int-vmx1

4. start VM with new configuration
    a. virsh define vfp-vmx1.xml
    b. virsh define vcp-vmx1.xml
    c. virsh net-define br-int-vmx1.xml
    d. virsh net-start br-int-vmx1
    e. virsh start vfp-vmx1
    f. virsh start vcp-vmx1

Chapter 7. troubleshooting installation script

7.1. ixgbe compilation error

```
ping@trinity:/images/vmx_20151102.0$ sudo ./vmx.sh -lvf --install
......

[OK]
Check IXGBE drivers.............................
[Command] cd /virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src

[Command] rm -f ixgbe.ko

[Command] make install
make -C /lib/modules/3.19.0-25-generic/build
SUBDIRS=/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src modules
make[1]: Entering directory `/usr/src/linux-headers-3.19.0-25-generic'
  CC [M]  /virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.o
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c: In
function ‘ixgbe_service_event_complete’:
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c:339:2:
error: implicit declaration of function ‘smp_mb__before_clear_bit’ [-Werror=implicit-
function-declaration]
  smp_mb__before_clear_bit();
  ^
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c: In
function ‘ixgbe_rx_hash’:
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c:923:6:
error: ‘struct sk_buff’ has no member named ‘rxhash’
   skb->rxhash = le32_to_cpu(rx_desc->wb.lower.hi_dword.rss);
      ^
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c: In
function ‘ixgbe_del_mac_filter’:
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c:4702:3:
error: implicit declaration of function ‘compare_ether_addr’ [-Werror=implicit-
function-declaration]
   if (!compare_ether_addr(addr, adapter->mac_table[i].addr) &&
   ^
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c: In
function ‘ixgbe_ndo_bridge_getlink’:
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c:8800:2:
error: too few arguments to function ‘ndo_dflt_bridge_getlink’
  return ndo_dflt_bridge_getlink(skb, pid, seq, dev, mode);
  ^
In file included from include/net/dst.h:13:0,
                 from include/net/sock.h:68,
                 from include/linux/tcp.h:22,
```

```
                 from /virtualization/images/vmx_20151102.0/drivers/ixgbe-
3.19.1/src/ixgbe_main.c:40:
include/linux/rtnetlink.h:110:12: note: declared here
 extern int ndo_dflt_bridge_getlink(struct sk_buff *skb, u32 pid, u32 seq,
            ^
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c: At top
level:
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c:8830:2:
error: unknown field ‘ndo_set_vf_tx_rate’ specified in initializer
  .ndo_set_vf_tx_rate = ixgbe_ndo_set_vf_bw,
  ^
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c:8830:2:
warning: initialization from incompatible pointer type [enabled by default]
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c:8830:2:
warning: (near initialization for ‘ixgbe_netdev_ops.ndo_set_vf_rate’) [enabled by
default]
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c:8872:2:
warning: initialization from incompatible pointer type [enabled by default]
  .ndo_fdb_add  = ixgbe_ndo_fdb_add,
  ^
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c:8872:2:
warning: (near initialization for ‘ixgbe_netdev_ops.ndo_fdb_add’) [enabled by default]
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c: In
function ‘ixgbe_ndo_bridge_getlink’:
/virtualization/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.c:8801:1:
warning: control reaches end of non-void function [-Wreturn-type]
 }
 ^
cc1: some warnings being treated as errors
make[2]: *** [/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src/ixgbe_main.o] Error 1
make[1]: *** [_module_/images/vmx_20151102.0/drivers/ixgbe-3.19.1/src] Error 2
make[1]: Leaving directory `/usr/src/linux-headers-3.19.0-25-generic'
make: *** [default] Error 2
[Failed]
Log
file......................................./images/vmx_20151102.0/build/vmx1/logs/v
mx_1447967843.log
================================================
 Aborted!. 1 error(s) and 0 warning(s)
================================================
ping@trinity:/images/vmx_20151102.0$
ping@trinity:/images/vmx_20151102.0$
ping@trinity:/images/vmx_20151102.0$
```

7.1.1. analysis and solution

This is ixgbe kernel driver module compilation error, and the cause is a "wrong" kernel version.

Currently the solution is to change the host ubuntu kernel to 3.13, see section install required linux kernel.

7.2. 82599 NIC not recognized

```
......
[OK]
Setup huge pages to 32768.........................
[Command] echo 32768 > /proc/sys/vm/nr_overcommit_hugepages

[Command] mkdir /HugePage_vPFE

[Command] mount | grep "HugePage_vPFE"

[Command] mount -t hugetlbfs hugetlbfs /HugePage_vPFE
[OK]

[Command] cat /etc/apparmor.d/abstractions/libvirt-qemu  | grep HugePage_vPFE

[Command] echo "owner \"/HugePage_vPFE/libvirt/qemu/**\" rw," >>
/etc/apparmor.d/abstractions/libvirt-qemu

[Command] modprobe pci-stub

[Command] echo "8086 10ed" > /sys/bus/pci/drivers/pci-stub/new_id

[Command] > /sys/bus/pci/drivers/pci-stub/bind
Number of Intel 82599 NICs.......................0
Minimum Intel 82599 NICs available.............No
================================================
    Aborted!
================================================
```

7.2.1. analysis and solution

the script does not find any of the specific NICs variants it was looking for:

- 82599 ES
- 82599 EB
- 82599 EN

and so it aborted.

```
66 vmx_system_setup_ixgbe()
67 {
68     nic_count=$(lspci | grep 82599 | grep 'ES\|EB\|EN' | wc -l)      #<------
69     vmx_echo_textval "Number of Intel 82599 NICs" "$nic_count"
70     if [ $nic_count -eq 0 ];
71     then
72         vmx_echo_textval_red "Minimum Intel 82599 NICs available" "No"
73         vmx_echo_summary_banner "Aborted!"
74         exit 2
75     fi
76     vmx_echo_text "Configuring Intel 82599 Adapters for SRIOV"
77     cmd="rmmod ixgbevf"
78     vmx_exec_cmd -cmd "$cmd"
79     cmd="rmmod ixgbe"
80     vmx_exec_cmd -cmd "$cmd"
81     max_vfs_str="1"
82     nic_count=$(expr $nic_count - 1)
83     while [ $nic_count -gt 0 ];
84     do
85         max_vfs_str="$max_vfs_str,1"
86         nic_count=$(expr $nic_count - 1)
87     done
88     cmd="modprobe ixgbe max_vfs=$max_vfs_str"
89     vmx_exec_cmd -cmd "$cmd"
90     cmd="modprobe ixgbevf"
91     vmx_exec_cmd -cmd "$cmd"
92     cmd="modprobe tun"
93     vmx_exec_cmd -cmd "$cmd"
94     vmx_echo "[OK]"
95     cmd_descr="Number of Virtual Functions created"
96     cmd="lspci |grep 82599 | grep \"Virtual Function\" | wc -l"
97     vmx_exec_cmd -cmd "$cmd" -cmd_descr "$cmd_descr"
98 }
```

The solution is to modify the installation script to ignore the `NS BS EN` keyword when looking for `82599` NIC

7.3. hyperthread

```
================================================
    System Setup Completed
================================================

[Command] brctl show br-ext | grep em1
can't get info No such device
Get Management Address of em1.....................
[Command] expr "10.85.4.17" != ""
1
[OK]
Generate libvirt files............................
[Command] python /virtualization/images/vmx_20151102.0/scripts/common/vmx_configure.py
/virtualization/images/vmx_20151102.0/config/vmx.conf
Handling Host config
Initializing interface names for vmx1
Model=FPC
10.85.4.17
255.255.255.128
handling bridge config
Handling Routing Engine params
Handling Forwarding Engine params
['0', '1', '2', '3', '4', '5', '6', '7']
Core list HTO:
['0', '1', '2', '3', '4', '5', '6', '7']
Core list HT1:
[]
0
[Failed]
Traceback (most recent call last):
  File "/images/vmx_20151102.0/scripts/common/vmx_configure.py", line 673, in <module>
    node_list[index].cpu_list[2*corenum + 1] = core_list_ht1[corenum]
IndexError: list index out of range
Log
file........................................../images/vmx_20151102.0/build/vmx1/logs/v
mx_1448050510.log
================================================
 Aborted!. 1 error(s) and 0 warning(s)
================================================
```

7.3.1. analysis and solution:

starting for 15.x VMX start to support flow-cache which by design requires hyperthreading feature.

this can be fixed by either:

- enable hyperthread
- change script to ignore hyperthreading checking

```
2134     #core_list_ht1 = core_list.cpu_list[num_physical_cores_per_node :
num_physical_cores_per_node*2]
2135
2136     print "Core list HTO: "
2137     print core_list_ht0
2138     #print "Core list HT1: "       ①
2139     #print core_list_ht1           ①
2140
2141     for corenum in range(num_physical_cores_per_node):
2142
2143         print corenum
2144         #node_list[index].cpu_list[2*corenum] = core_list_ht0[corenum]    ②
2145         node_list[index].cpu_list[corenum] = core_list_ht0[corenum]       ②
2146         #node_list[index].cpu_list[2*corenum + 1] = core_list_ht1[corenum] ②
2147
2148         #for corenum in range(len(node_list)):
2149         #       node_list[index].cpu_list[corenum] =
2150     print node_list[index].cpu_list
```

① comment these lines

② disable hyperthreading check

Part 2: VMX verification

Chapter 8. VCP/VFP guest VM overview

After installation of VMX, at least two VMs should have been created:

- VM for virtual forwarding plane of VMX (VFP/vFPC)
- VM for virtual control plane of VMX (VCP/vRE)

The VFP VM runs the (`yocto` based) virtual Trio forwarding plane software and the VCP VM runs (`Freebsd` based) Junos OS.

This is demonstrated in the VMX architecture:

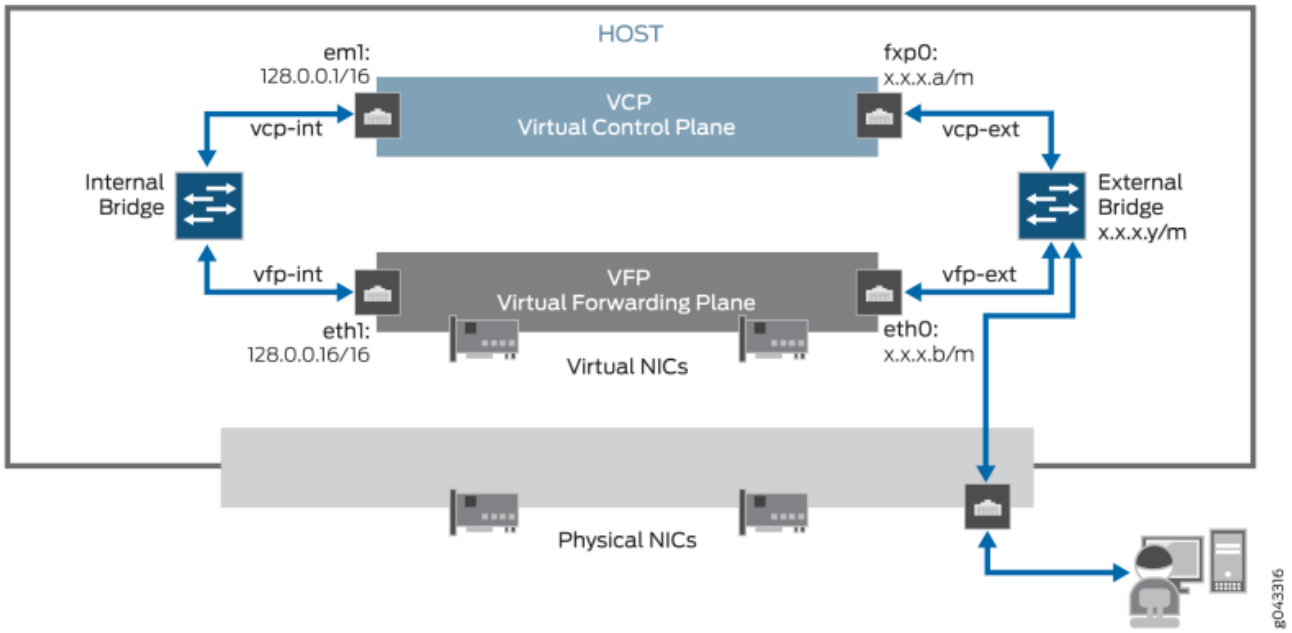


Figure 2. architecture of VMX instance

These VMs should be in "running" state, as can be listed with libvirt `virsh` command below:

```
ping@trinity:~$ sudo virsh list
[sudo] password for ping:
 Id    Name                           State
----------------------------------------------------
 2     vcp-vmx1                       running
 3     vfp-vmx1                       running
```

The next thing we want to verify is the images these VM were built from:

```
ping@trinity:~$ sudo virsh domblklist 2
Target     Source
------------------------------------------------
hda        /virtualization/images/vmx_20151102.0/build/vmx1/images/jinstall64-vmx-
15.1F-20151104.0-domestic.img
hdb        /virtualization/images/vmx_20151102.0/build/vmx1/images/vmxhdd.img
sda        /virtualization/images/vmx_20151102.0/images/metadata_usb.img
```

```
ping@trinity:~$ sudo virsh domblklist 3
Target     Source
------------------------------------------------
hda        /virtualization/images/vmx_20151102.0/build/vmx1/images/vFPC-20151102.img
```

NOTE: the libvirt `virsh` utility is a very powerful frontend toolset that can be used to manage the VMs under its control [3: in the same host there can be some other VMs that may not be spinned up by libvirt, in that case those VM won't be under control of libvirt]. There are a lot more other data can be gathered simply from libvirt `virsh` tool, without the need to login to guest VMs. see VM managent - virsh section for more details on this.

Chapter 9. Login to the VNC console

Right after a fresh VMX installation, the guest VCP/VFP VM will come up with no configs. So the initial configuration (mgmt IP, GW, login, etc) needs to be done via a "console" connection. Previously we demonstrated how to login to the vRE and vPFE `console` via `telnet`, which is also a convenient method to collect the guest VM booting messages in the case that IP-based management session is not available. Another way to acquire "console" of a guest VM is via the built-in `VNC` support provided by KVM.

locate the console/VNC access ports

A quick way to find out tcp port for console connection is to check `netstat`:

```
ping@trinity: $ sudo netstat -lntp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address  Foreign Address State   PID/Program name
tcp    0     0   127.0.0.1:5900    0.0.0.0:*       LISTEN  99035/qemu-system-x ①
tcp    0     0   127.0.0.1:5901    0.0.0.0:*       LISTEN  99192/qemu-system-x ①
tcp    0     0   127.0.0.1:8816    0.0.0.0:*       LISTEN  99035/qemu-system-x ②
tcp    0     0   127.0.0.1:8817    0.0.0.0:*       LISTEN  99192/qemu-system-x ②
tcp    0     0   10.85.4.17:53     0.0.0.0:*       LISTEN  98162/dnsmasq       ③
tcp    0     0   192.168.122.1:53  0.0.0.0:*       LISTEN  1731/dnsmasq        ③
tcp    0     0   0.0.0.0:22        0.0.0.0:*       LISTEN  1438/sshd
tcp    0     0   127.0.0.1:6010    0.0.0.0:*       LISTEN  2677/1
tcp    0     0   127.0.0.1:6011    0.0.0.0:*       LISTEN  10570/2
tcp    0     0   127.0.0.1:6012    0.0.0.0:*       LISTEN  10636/3
tcp    0     0   127.0.0.1:6013    0.0.0.0:*       LISTEN  14223/4
```

① VNC service port

② console port

③ DNS service port [4: the dns service here is not of our concern at this time, knowning we also have these "extra" services enabled does not harm anyway]

VNC ports can also be acquired via `virsh` command:

```
ping@trinity:~$ sudo virsh vncdisplay 2
127.0.0.1:0

ping@trinity:~$ sudo virsh vncdisplay 3
127.0.0.1:1
```

This indicates VNC port 5900 and 5901 for VCP and VFP respectively, same as shown in previous `netstat` command output.

Now VNC GUI can be accessed via legacy `vncviewer` or `virt-viewer` tools, from the server's GUI.

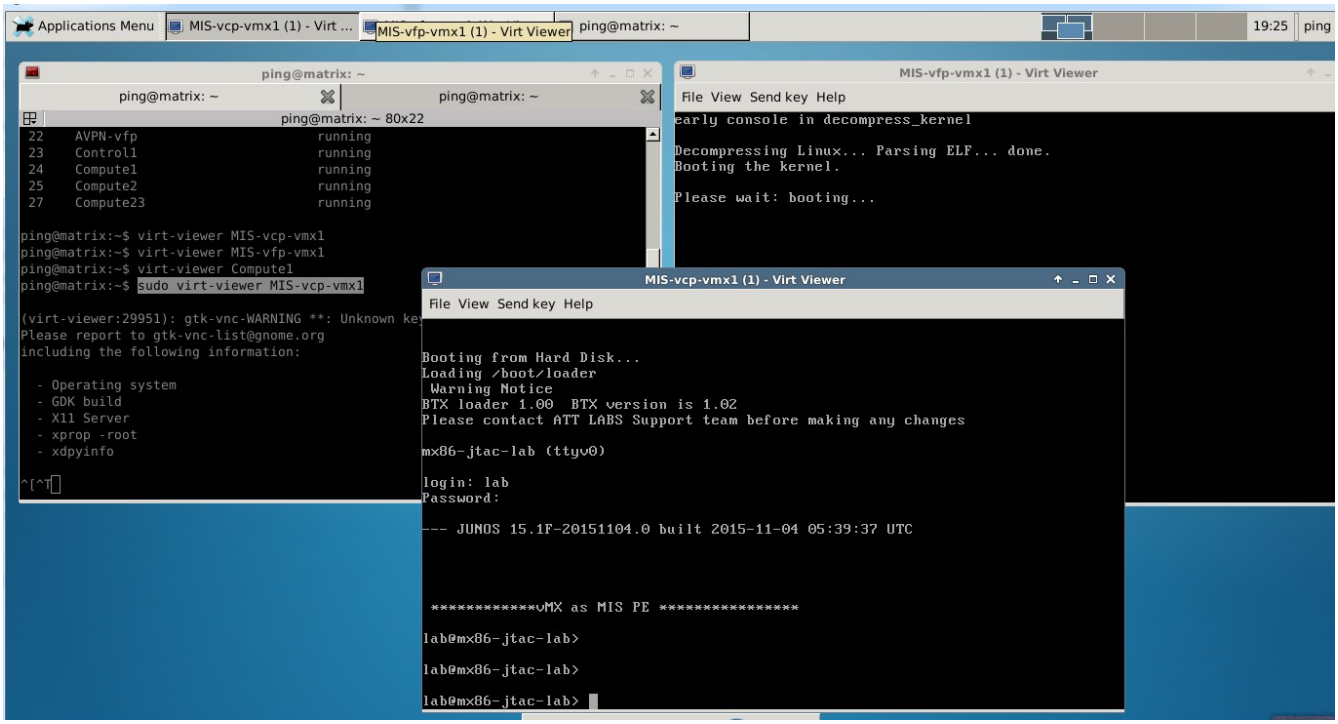


Figure 3. example of using libvirt virt-viewer to access VNC console

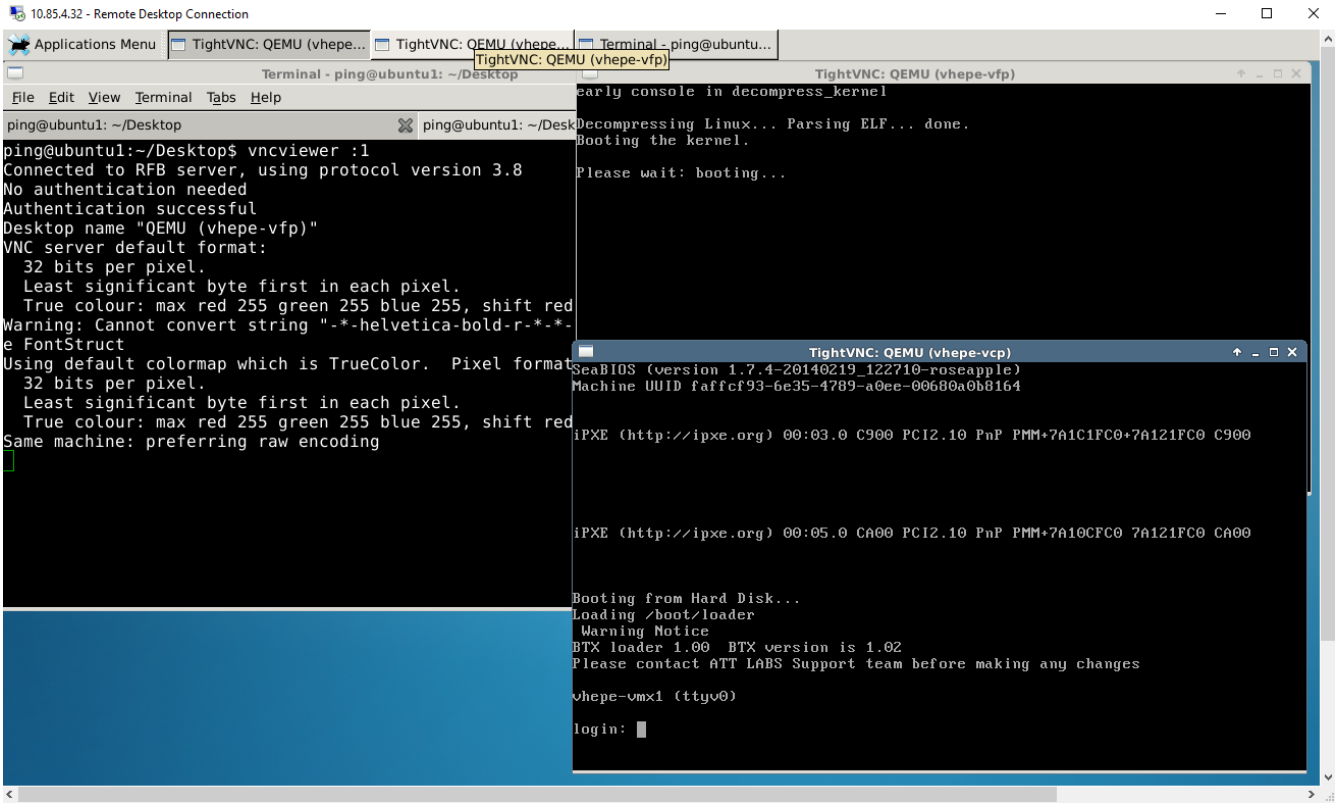


Figure 4. example of using linux vncviewer to access VNC console

Chapter 10. ixgbe driver and ixgbe-driven interfaces

With SR-IOV version of VMX installation, Juniper modified ixgbe driver will be in use:

- ixgbe kernel driver version is 3.19.1 (Juniper modified)
- some more parameters are now supported than the default driver coming with linux kernel
- ixgbevf driver kernel module is now loaded to support SR-IOV Virtual Function

```
ping@trinity:~$ modinfo ixgbe
filename:       /lib/modules/3.13.0-32-
generic/kernel/drivers/net/ethernet/intel/ixgbe/ixgbe.ko
version:        3.19.1      #<------
license:        GPL
description:    Intel(R) 10 Gigabit PCI Express Network Driver
author:         Intel Corporation, <linux.nics@intel.com>
srcversion:     B97B1E7CF79A25F5E4D7B96
alias:          pci:v00008086d00001560sv*sd*bc*sc*i*
alias:          pci:v00008086d00001558sv*sd*bc*sc*i*
alias:          pci:v00008086d0000154Asv*sd*bc*sc*i*
alias:          pci:v00008086d00001557sv*sd*bc*sc*i*
alias:          pci:v00008086d0000154Fsv*sd*bc*sc*i*
alias:          pci:v00008086d0000154Dsv*sd*bc*sc*i*
alias:          pci:v00008086d00001528sv*sd*bc*sc*i*
alias:          pci:v00008086d000010F8sv*sd*bc*sc*i*
alias:          pci:v00008086d0000151Csv*sd*bc*sc*i*
alias:          pci:v00008086d00001529sv*sd*bc*sc*i*
alias:          pci:v00008086d0000152Asv*sd*bc*sc*i*
alias:          pci:v00008086d000010F9sv*sd*bc*sc*i*
alias:          pci:v00008086d00001514sv*sd*bc*sc*i*
alias:          pci:v00008086d00001507sv*sd*bc*sc*i*
alias:          pci:v00008086d000010FBsv*sd*bc*sc*i*
alias:          pci:v00008086d00001517sv*sd*bc*sc*i*
alias:          pci:v00008086d000010FCsv*sd*bc*sc*i*
alias:          pci:v00008086d000010F7sv*sd*bc*sc*i*
alias:          pci:v00008086d00001508sv*sd*bc*sc*i*
alias:          pci:v00008086d000010DBsv*sd*bc*sc*i*
alias:          pci:v00008086d000010F4sv*sd*bc*sc*i*
alias:          pci:v00008086d000010E1sv*sd*bc*sc*i*
alias:          pci:v00008086d000010F1sv*sd*bc*sc*i*
alias:          pci:v00008086d000010ECsv*sd*bc*sc*i*
alias:          pci:v00008086d000010DDsv*sd*bc*sc*i*
alias:          pci:v00008086d0000150Bsv*sd*bc*sc*i*
alias:          pci:v00008086d000010C8sv*sd*bc*sc*i*
alias:          pci:v00008086d000010C7sv*sd*bc*sc*i*
alias:          pci:v00008086d000010C6sv*sd*bc*sc*i*
alias:          pci:v00008086d000010B6sv*sd*bc*sc*i*
depends:        dca
```

```
vermagic:       3.13.0-32-generic SMP mod_unload modversions
parm:           InterruptType:Change Interrupt Mode (0=Legacy, 1=MSI, 2=MSI-X),
default IntMode (deprecated) (array of int)
parm:           IntMode:Change Interrupt Mode (0=Legacy, 1=MSI, 2=MSI-X), default 2
(array of int)
parm:           MQ:Disable or enable Multiple Queues, default 1 (array of int)
parm:           DCA:Disable or enable Direct Cache Access, 0=disabled, 1=descriptor
only, 2=descriptor and data (array of int)
parm:           RSS:Number of Receive-Side Scaling Descriptor Queues, default 0=number
of cpus (array of int)
parm:           VMDQ:Number of Virtual Machine Device Queues: 0/1 = disable, 2-16
enable (default=8) (array of int)
parm:           max_vfs:Number of Virtual Functions: 0 = disable (default), 1-63 =
enable this many VFs (array of int)
parm:           L2LBen:L2 Loopback Enable: 0 = disable, 1 = enable (default) (array of
int)
parm:           InterruptThrottleRate:Maximum interrupts per second, per vector,
(0,1,956-488281), default 1 (array of int)
parm:           LLIPort:Low Latency Interrupt TCP Port (0-65535) (array of int)
parm:           LLIPush:Low Latency Interrupt on TCP Push flag (0,1) (array of int)
parm:           LLISize:Low Latency Interrupt on Packet Size (0-1500) (array of int)
parm:           LLIEType:Low Latency Interrupt Ethernet Protocol Type (array of int)
parm:           LLIVLANP:Low Latency Interrupt on VLAN priority threshold (array of
int)
parm:           FdirPballoc:Flow Director packet buffer allocation level:
                        1 = 8k hash filters or 2k perfect filters
                        2 = 16k hash filters or 4k perfect filters
                        3 = 32k hash filters or 8k perfect filters (array of int)
parm:           AtrSampleRate:Software ATR Tx packet sample rate (array of int)
parm:           LRO:Large Receive Offload (0,1), default 1 = on (array of int)
parm:           allow_unsupported_sfp:Allow unsupported and untested SFP+ modules on
82599 based adapters, default 0 = Disable (array of int)
```

The ixgbe driver used here is the Juniper modified version in order to support the ability of accepting ingress multicast packets arriving in a VF. So it's different than what is available from Intel website, even though both may show same version number.

```
root@ubuntu1:~# modinfo ixgbevf
filename:       /lib/modules/3.13.0-32-
generic/kernel/drivers/net/ethernet/intel/ixgbevf/ixgbevf.ko
version:        2.11.3-k
license:        GPL
description:    Intel(R) 82599 Virtual Function Driver
author:         Intel Corporation, <linux.nics@intel.com>
srcversion:     AE2D8A25951B508611E943D
alias:          pci:v00008086d00001515sv*sd*bc*sc*i*
alias:          pci:v00008086d000010EDsv*sd*bc*sc*i*
depends:
intree:         Y
vermagic:       3.13.0-32-generic SMP mod_unload modversions
signer:         Magrathea: Glacier signing key
sig_key:        5E:3C:0F:9C:A6:E3:65:43:53:5F:A2:BB:5B:70:9E:84:F1:6D:A7:C7
sig_hashalgo:   sha512
parm:           debug:Debug level (0=none,...,16=all) (int)
```

To verify interfaces status from a linux server, the most commonly used commands are:

- `ifconfig` command: "legacy" system administration utility in Unix-like OS to manipulate or show interfaces
- `ip` command : relatively new commands, show / manipulate interface, routing, devices, policy routing and tunnels, etc. much more features-rich than the traditional `ifconfig` tool.
- `ethtool` command: to query or control network driver and low level hardware settings, mostly used to polling L1 physical layer information of an interface (e.g PCI address)

in this section we'll use `ifconfig` and `ip link show` to list the physical ports, VFs and bridges.

10.1. legacy `ifconfig` command

This is what it looks like after one VMX instance was built and brought up to running status:

```
ping@trinity:/virtualization/images/vmx_20151102.0$ ifconfig -a

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:37 errors:0 dropped:0 overruns:0 frame:0
          TX packets:37 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:6468 (6.4 KB)  TX bytes:6468 (6.4 KB)

br-ext    Link encap:Ethernet  HWaddr 38:ea:a7:37:7c:54                \     ①
          inet addr:10.85.4.17  Bcast:10.85.4.127  Mask:255.255.255.128  |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1             |
          RX packets:661 errors:0 dropped:0 overruns:0 frame:0           |
```

```
          TX packets:584 errors:0 dropped:0 overruns:0 carrier:0          |
          collisions:0 txqueuelen:0                                       |
          RX bytes:36305 (36.3 KB)  TX bytes:77528 (77.5 KB)              |
                                                                          |
br-ext-nic Link encap:Ethernet  HWaddr 52:54:00:9f:a0:77                  |    ②
          BROADCAST MULTICAST  MTU:1500  Metric:1                         |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0              |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0            |
          collisions:0 txqueuelen:500                                    \
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                          X br-ext
bridge
                                                                          |  and I/Fs
vcp_ext-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:01:01                |    ③
          inet6 addr: fe80::fc04:17ff:fe01:101/64 Scope:Link              |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1              |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0              |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0            |
          collisions:0 txqueuelen:500                                     |
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                          |
                                                                          |
vfp_ext-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:01:02                |    ④
          inet6 addr: fe80::fc04:17ff:fe01:102/64 Scope:Link              |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1              |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0              |
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0           |
          collisions:0 txqueuelen:500                                     |
          RX bytes:0 (0.0 B)  TX bytes:6934 (6.9 KB)                     /

br-int-vmx1 Link encap:Ethernet  HWaddr 52:54:00:ad:64:15                 \    ⑤
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1               |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0               |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0             |
          collisions:0 txqueuelen:0                                        |
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                           |
                                                                           |    ⑥
br-int-vmx1-nic Link encap:Ethernet  HWaddr 52:54:00:ad:64:15              |
          BROADCAST MULTICAST  MTU:1500  Metric:1                          |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0               |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0             |
          collisions:0 txqueuelen:500                                     \
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                           X
                                                                           |
vcp_int-vmx1 Link encap:Ethernet  HWaddr fe:54:00:84:52:fb                 |    ⑦
          inet6 addr: fe80::fc54:ff:fe84:52fb/64 Scope:Link                |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1               |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0               |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0             |
          collisions:0 txqueuelen:500                                      |
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                           |
                                                                           |
vfp_int-vmx1 Link encap:Ethernet  HWaddr fe:54:00:c0:ff:1f                 |    ⑧
```

```
          inet6 addr: fe80::fc54:ff:fec0:ff1f/64 Scope:Link                |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1               |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0               |
          TX packets:22 errors:0 dropped:0 overruns:0 carrier:0            |
          collisions:0 txqueuelen:500                                      |
          RX bytes:0 (0.0 B)  TX bytes:1376 (1.3 KB)                      /

virbr0    Link encap:Ethernet  HWaddr 2a:e1:60:a8:87:40                    \
          inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0 |
          UP BROADCAST MULTICAST  MTU:1500  Metric:1                        | ⑨
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0                |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0              |
          collisions:0 txqueuelen:0                                         |
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                           /

em1       Link encap:Ethernet  HWaddr 38:ea:a7:37:7c:54          \
          inet6 addr: fe80::3aea:a7ff:fe37:7c54/64 Scope:Link     \
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1       |
          RX packets:1167 errors:0 dropped:1 overruns:0 frame:0    |
          TX packets:1071 errors:0 dropped:0 overruns:0 carrier:0  |
          collisions:0 txqueuelen:1000                             |
          RX bytes:80815 (80.8 KB)  TX bytes:129388 (129.3 KB)     |
                                                                   |
em2       Link encap:Ethernet  HWaddr 38:ea:a7:37:7c:55            |
          BROADCAST MULTICAST  MTU:1500  Metric:1                  |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0       |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0     |
          collisions:0 txqueuelen:1000                             |
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                   |
                                                                   |
em9       Link encap:Ethernet  HWaddr 38:ea:a7:37:7b:d0            |
          BROADCAST MULTICAST  MTU:1500  Metric:1                  |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0       |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0     |
          collisions:0 txqueuelen:1000                             |
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                   |
                                                                   |
em10      Link encap:Ethernet  HWaddr 38:ea:a7:37:7b:d1            |
          BROADCAST MULTICAST  MTU:1500  Metric:1                  |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0       |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0     |
          collisions:0 txqueuelen:1000                             |
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                   |
                                                                  \
p2p1      Link encap:Ethernet  HWaddr 38:ea:a7:17:65:a0             X ⑩
          inet6 addr: fe80::3aea:a7ff:fe17:65a0/64 Scope:Link     /
          UP BROADCAST RUNNING PROMISC ALLMULTI MULTICAST  MTU:2000| Metric:1
          RX packets:9 errors:0 dropped:0 overruns:0 frame:0       |
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0     |
          collisions:0 txqueuelen:1000                             |
          RX bytes:810 (810.0 B)  TX bytes:558 (558.0 B)           |
```

```
                                                                      |
p2p2      Link encap:Ethernet  HWaddr 38:ea:a7:17:65:a1               |
          BROADCAST MULTICAST  MTU:1500  Metric:1                     |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0          |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0        |
          collisions:0 txqueuelen:1000                                |
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                      |
                                                                      |
p3p1      Link encap:Ethernet  HWaddr 38:ea:a7:17:65:84               |
          inet6 addr: fe80::3aea:a7ff:fe17:6584/64 Scope:Link         |
          UP BROADCAST RUNNING PROMISC ALLMULTI MULTICAST  MTU:2000| Metric:1
          RX packets:1 errors:0 dropped:1 overruns:0 frame:0          |
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0        |
          collisions:0 txqueuelen:1000                                |
          RX bytes:301 (301.0 B)  TX bytes:558 (558.0 B)              |
                                                                      |
p3p2      Link encap:Ethernet  HWaddr 38:ea:a7:17:65:85               |
          BROADCAST MULTICAST  MTU:1500  Metric:1                     |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0          |
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0        |
          collisions:0 txqueuelen:1000                               /
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)                    /
```

① the "external" bridge that bridges vRE (fxp0) and vPFE (ext) mgmt I/F to the mgmt I/F of the "host" or server (em1)

② br-ext-nic interface

③ `vcp_ext-vmx1`: external tap interface for `vcp-vmx1` fxp0 interface

④ `vfp_ext-vmx1`: external tap inferface for `vfp-vmx1` ext interface

⑤ the "internal" bridge, that connects vRE (em1) and vPFE (eth1) for the internal communications (software module download, IPC, PFE-RE host bound traffic, etc)

⑥ `br-int-nic` interface

⑦ internal tap interface for `vcp-vmx1` em1

⑧ internal tap interface for `vfp-vmx1` eth1

⑨ the default virtual network (VN), created when `libvirtd` daemon was first installed and started. missing of this interface may indicate something wrong with libvirt

the tap interface with "configured" MAC address

```
ping@trinity:/virtualization/images/vmx_20151102.0$ ifconfig -a | grep 04:17
vcp_ext-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:01:01      ①
          inet6 addr: fe80::fc:04:17:ff:fe01:101/64 Scope:Link
vfp_ext-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:01:02      ②
          inet6 addr: fe80::fc:04:17:ff:fe01:102/64 Scope:Link
```

① "peer" tap interface [1: interface showing in host, and peering with guest VM interface, e.g. the `vcp_ext-vmx1` interface is "peer interface" of fxp0 in VMX] for mgmt interface (fxp0) in VCP VM

② "peer" tap interface for mgmt interface (ext) in VFP VM

10.2. `ip` tool (SR-IOV)

comparing with legacy `ifconfig` command, the new `ip` tool is more powerful - In this example it prints the configured SR-IOV VF info which `ifconfig` does not provide, with much more interface properties.

```
ping@trinity:/virtualization/images/vmx_20151102.0$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group
default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
10: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode
DEFAULT group default
    link/ether 2a:e1:60:a8:87:40 brd ff:ff:ff:ff:ff:ff
27: em9: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
    link/ether 38:ea:a7:37:7b:d0 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
28: em10: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
    link/ether 38:ea:a7:37:7b:d1 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
29: p2p1: <BROADCAST,MULTICAST,ALLMULTI,PROMISC,UP,LOWER_UP> mtu 2000 qdisc mq state
UP mode DEFAULT group default qlen 1000
    link/ether 38:ea:a7:17:65:a0 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 02:04:17:01:02:02, tx rate 10000 (Mbps), spoof checking off, link-state
auto               ①
30: p2p2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
    link/ether 38:ea:a7:17:65:a1 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
31: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master br-ext state UP
mode DEFAULT group default qlen 1000
    link/ether 38:ea:a7:37:7c:54 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
32: em2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
    link/ether 38:ea:a7:37:7c:55 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
33: p3p1: <BROADCAST,MULTICAST,ALLMULTI,PROMISC,UP,LOWER_UP> mtu 2000 qdisc mq state
UP mode DEFAULT group default qlen 1000
    link/ether 38:ea:a7:17:65:84 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 02:04:17:01:02:01, tx rate 10000 (Mbps), spoof checking off, link-state
aut                ②
34: p3p2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
    link/ether 38:ea:a7:17:65:85 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
35: br-ext: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode
```

```
DEFAULT group default
    link/ether 38:ea:a7:37:7c:54 brd ff:ff:ff:ff:ff:ff
36: br-ext-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast master br-ext state
DOWN mode DEFAULT group default qlen 500
    link/ether 52:54:00:9f:a0:77 brd ff:ff:ff:ff:ff:ff
37: br-int-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
mode DEFAULT group default
    link/ether 52:54:00:ad:64:15 brd ff:ff:ff:ff:ff:ff
38: br-int-vmx1-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast master br-int-
vmx1 state DOWN mode DEFAULT group default qlen 500
    link/ether 52:54:00:ad:64:15 brd ff:ff:ff:ff:ff:ff
39: vcp_ext-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master
br-ext state UNKNOWN mode DEFAULT group default qlen 500
    link/ether fe:04:17:01:01:01 brd ff:ff:ff:ff:ff:ff
40: vcp_int-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master
br-int-vmx1 state UNKNOWN mode DEFAULT group default qlen 500
    link/ether fe:54:00:84:52:fb brd ff:ff:ff:ff:ff:ff
41: vfp_ext-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master
br-ext state UNKNOWN mode DEFAULT group default qlen 500
    link/ether fe:04:17:01:01:02 brd ff:ff:ff:ff:ff:ff
42: vfp_int-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master
br-int-vmx1 state UNKNOWN mode DEFAULT group default qlen 500
    link/ether fe:54:00:c0:ff:1f brd ff:ff:ff:ff:ff:ff
```

the tap interface and VF with "configured" MAC address

```
29: p2p1: <BROADCAST,MULTICAST,ALLMULTI,PROMISC,UP,LOWER_UP> mtu 2000 qdisc mq state
UP mode DEFAULT group default qlen 1000
    link/ether 38:ea:a7:17:65:a0 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 02:04:17:01:02:02, tx rate 10000 (Mbps), spoof checking off, link-state
auto               ①
33: p3p1: <BROADCAST,MULTICAST,ALLMULTI,PROMISC,UP,LOWER_UP> mtu 2000 qdisc mq state
UP mode DEFAULT group default qlen 1000
    link/ether 38:ea:a7:17:65:84 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 02:04:17:01:02:01, tx rate 10000 (Mbps), spoof checking off, link-state
auto               ②
39: vcp_ext-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master
br-ext state UNKNOWN mode DEFAULT group default qlen 500
    link/ether fe:04:17:01:01:01 brd ff:ff:ff:ff:ff:ff
41: vfp_ext-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master
br-ext state UNKNOWN mode DEFAULT group default qlen 500
    link/ether fe:04:17:01:01:02 brd ff:ff:ff:ff:ff:ff
```

① p2p1 VF 0 L1/L2 info, this will map to the VMX router `ge-0/0/1` interface

② p3p1 VF 0 L1/L2 info, this will map to the VMX router `ge-0/0/0` interface

10.3. host interfaces (virtio)

the list of interfaces after installation of an `virtio` version of VMX looks very similiar to the ones

after SR-IOV verions of VMX installation, but with one exception - Besides external and internal bridges and the associated tap interfaces for management, now we can see **2 more tap interfaces generated by virtio**.

```
ge-0.0.0-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:02:01     \
          inet6 addr: fe80::fc04:17ff:fe01:201/64 Scope:Link     |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1     |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0     |
          TX packets:401 errors:0 dropped:0 overruns:0 carrier:0 |
          collisions:0 txqueuelen:500                            \
          RX bytes:0 (0.0 B)  TX bytes:21084 (21.0 KB)            X ①
                                                                 |
ge-0.0.1-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:02:02      |
          inet6 addr: fe80::fc04:17ff:fe01:202/64 Scope:Link     |
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1     |
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0     |
          TX packets:401 errors:0 dropped:0 overruns:0 carrier:0 |
          collisions:0 txqueuelen:500                            /
          RX bytes:0 (0.0 B)  TX bytes:21084 (21.0 KB)          /
```

① virtio interfaces

"virtio interface" is tap interface

```
ping@trinity:~$ sudo ethtool -i ge-0.0.0-vmx1
driver: tun
version: 1.6
firmware-version:
bus-info: tap   #<------
supports-statistics: no
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: no
```

MAC address (virtio)

```
ping@trinity:/virtualization/images/vmx_20151102.0$ ifconfig -a | grep -i 04:17
ge-0.0.0-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:02:01
          inet6 addr: fe80::fc04:17ff:fe01:201/64 Scope:Link
ge-0.0.1-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:02:02
          inet6 addr: fe80::fc04:17ff:fe01:202/64 Scope:Link
vcp_ext-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:01:01
          inet6 addr: fe80::fc04:17ff:fe01:101/64 Scope:Link
vfp_ext-vmx1 Link encap:Ethernet  HWaddr fe:04:17:01:01:02
          inet6 addr: fe80::fc04:17ff:fe01:102/64 Scope:Link
virbr0    Link encap:Ethernet  HWaddr fe:04:17:01:02:01
```

10.4. VMX fxp0 and em1 interface

In a physical MX box, `fxp0` is used for the external Ethernet port on the RE, while `em0`/`em1` (previously `fxp1`/`fxp2` in some old platforms) are internal NICs on the RE that cross-connected to the internal 24x1GE switch on each SCB.

With a up and running VMX , we can now look at the `fxp0` and `em1` interface, as if this is a physical MX router - with exactly the same Junos command.

`fxp0` emulates the same physical `fxp0` NIC in MX RE board, and is still used for the same purpose - serving managment connection from external device.

VCP VM management interface - fxp0 (`external` interface)

```
root@vmx1> show interfaces fxp0
Physical interface: fxp0, Enabled, Physical link is Up
  Interface index: 8, SNMP ifIndex: 1
  Type: Ethernet, Link-level type: Ethernet, MTU: 1514
  Device flags   : Present Running
  Interface flags: SNMP-Traps
  Current address: 02:04:17:01:01:01, Hardware address: 02:04:17:01:01:01
  Last flapped   : 2015-12-01 03:03:56 UTC (00:01:20 ago)
    Input packets : 384
    Output packets: 2

  Logical interface fxp0.0 (Index 4) (SNMP ifIndex 13)
    Flags: Up SNMP-Traps 0x4000000 Encapsulation: ENET2
    Input packets : 169
    Output packets: 2
    Protocol inet, MTU: 1500
      Flags: Sendbcast-pkt-to-re, Is-Primary
      Addresses, Flags: Is-Default Is-Preferred Is-Primary
        Destination: 10.85.4.0/25, Local: 10.85.4.105, Broadcast: 10.85.4.127
```

Although there is no real `CB` or `SCB` board in VMX, the interface `em1` remains - it is now connecting directly to the only FPC - VCP VM. As in MX router it can be viewed by the same `show interface` CLI:

VCP VM interface - em1 (`internal` interface connecting to VFP only)

```
root# run show interfaces em1
Physical interface: em1, Enabled, Physical link is Up
  Interface index: 9, SNMP ifIndex: 23
  Type: Ethernet, Link-level type: Ethernet, MTU: 1514, Speed: 1000mbps
  Device flags   : Present Running
  Interface flags: SNMP-Traps
  Link type      : Full-Duplex
  Current address: 52:54:00:a8:60:36, Hardware address: 52:54:00:a8:60:36
  Last flapped   : Never
    Input packets : 41993
    Output packets: 41520

  Logical interface em1.0 (Index 3) (SNMP ifIndex 24)
    Flags: Up SNMP-Traps 0x4000000 Encapsulation: ENET2
    Input packets : 41993
    Output packets: 41520
    Protocol inet, MTU: 1500
      Flags: Is-Primary
      Addresses, Flags: Is-Preferred
        Destination: 10/8, Local: 10.0.0.4, Broadcast: 10.255.255.255
      Addresses, Flags: Preferred Kernel Is-Preferred
        Destination: 128/2, Local: 128.0.0.1, Broadcast: 191.255.255.255
      Addresses, Flags: Primary Is-Default Is-Primary
        Destination: 128/2, Local: 128.0.0.4, Broadcast: 191.255.255.255
    Protocol inet6, MTU: 1500
    Max nh cache: 75000, New hold nh limit: 75000, Curr nh cnt: 0,
    Curr new hold cnt: 0, NH drop cnt: 0
      Flags: Is-Primary
      Addresses, Flags: Is-Preferred
        Destination: fe80::/64, Local: fe80::5254:ff:fea8:6036
      Addresses, Flags: Is-Default Is-Preferred Is-Primary
        Destination: fec0::/64, Local: fec0::a:0:0:4
    Protocol tnp, MTU: 1500
      Flags: Primary, Is-Primary
      Addresses
        Local: 0x4
```

10.5. `ge-x/y/z` interface

the `ge-x/y/z` interface, built from SR-IOV VF or virtio tag interface in the low level, represents the forwarding plane of VMX. The name `ge-` may not be acurate - it can be a 10GE or even 100GE capable port, depending on the type of (Intel) NIC card in use.

ge-0/0/0 interface from Junos

```
labroot> show interfaces ge-0/0/0
Physical interface: ge-0/0/0, Enabled, Physical link is Up
  Interface index: 139, SNMP ifIndex: 517
  Link-level type: Ethernet, MTU: 1514, MRU: 1522, LAN-PHY mode,
  Speed: 1000mbps, BPDU Error: None, MAC-REWRITE Error: None,
  Loopback: Disabled, Source filtering: Disabled, Flow control: Enabled
  Pad to minimum frame size: Disabled
  Device flags   : Present Running
  Interface flags: SNMP-Traps Internal: 0x4000
  Link flags     : None
  CoS queues     : 8 supported, 8 maximum usable queues
  Current address: 02:04:17:01:02:01, Hardware address: 02:04:17:01:02:01
  Last flapped   : 2015-11-25 07:14:41 UTC (00:00:08 ago)
  Input rate     : 0 bps (0 pps)
  Output rate    : 0 bps (0 pps)
  Active alarms  : None
  Active defects : None
  Interface transmit statistics: Disabled
```

Chapter 11. virtual networks/bridging (SR-IOV)

To list Virtual Networks (VN) generated by libvirt:

```
ping@trinity:~$ sudo virsh net-list
[sudo] password for ping:
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 br-ext               active     no            yes
 br-int-vmx1          active     no            yes
 default              active     yes           yes
```

These virtual networks were constructed via linux bridges:

VN `br-ext`

```
sudo virsh net-dumpxml br-ext

<network>
  <name>br-ext</name>
  <forward mode="route" />                                  ①
  <bridge delay="0" name="br-ext" stp="on" />               ②
  <mac address="52:54:00:9f:a0:77" />                       ③
  <ip address="10.85.4.17" netmask="255.255.255.128">       ③
    <dhcp>                                                  ④
      <host ip="10.85.4.105" mac="02:04:17:01:01:01" name="vcp-vmx1" /> ④
      <host ip="10.85.4.106" mac="02:04:17:01:01:02" name="vfp-vmx1" /> ④
    </dhcp>
  </ip>
</network>
```

① here the VN `br-ext` is in "route" mode, traffic to/from this VN will be "routed" to/from the host interface

② the host interface is a bridge with same name `br-ext`

③ the host bridge's MAC and IP info

④ IP address can be assigned based on MAC address from the libvirt built-in DHCP server

currently VMX mgmt interface implementation doesn't request IP address acquirement via DHCP. This may be changed in the future.

VN `br-int-vm1`

```
sudo virsh net-dumpxml br-int-vmx1

<network>
  <name>br-int-vmx1</name>
  <bridge delay="0" name="br-int-vmx1" stp="on" />      #<------
</network>
```

- the VN `br-int-vmx1` is an "isolated" network - there is no "forward" attribute as presenting in `br-ext` VN
- VN is implemented as a bridge with same name `br-int-vmx1`

the `default` *VN*

```
ping@trinity:~$ sudo virsh net-dumpxml default
<network connections='2'>
  <name>default</name>
  <uuid>35f8cee2-d217-4e1f-a4c0-e0e08c422a4e</uuid>
  <forward mode='nat'>                                    ②
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr0' stp='on' delay='0'/>              ①
  <ip address='192.168.122.1' netmask='255.255.255.0'>    ③
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'/> ④
    </dhcp>
  </ip>
</network>
```

① implemented as a bridge named `virbr0`,

② `nat` is used for traffic forwarding

③ IP address of `virbr0` bridge

④ IP address pool for the libvirt built-in DHCP server.

Essentially, the default VN emulates a typical CPE router, via which the guest VM can interact with external world without having to each has their own interfaces configured with an external IP and routing entry, just like a home router eliminates the need of having each home PC to acquire an external IP in order to access the Internet.

the VN to brige binding relationship can also be verified via this virsh command: `virsh net-info`

```
ping@trinity:~$ sudo virsh net-info default
Name:           default
UUID:           35f8cee2-d217-4e1f-a4c0-e0e08c422a4e
Active:         yes
Persistent:     yes
Autostart:      yes
Bridge:         virbr0

ping@trinity:~$ sudo virsh net-info br-ext
Name:           br-ext
UUID:           b7abcd87-114e-40f3-8b4e-e913135aae56
Active:         yes
Persistent:     yes
Autostart:      no
Bridge:         br-ext

ping@trinity:~$ sudo virsh net-info br-int-vmx1
Name:           br-int-vmx1
UUID:           36aa8f84-84e2-4bec-92cb-f2b41a4ed2f7
Active:         yes
Persistent:     yes
Autostart:      no
Bridge:         br-int-vmx1
```

The bridge to interface binding relationship is as following:

```
    ping@trinity:~$ brctl show
    bridge name bridge id       STP enabled interfaces
    br-ext      8000.38eaa7377c54   yes br-ext-nic
                                                em1
                                                vcp_ext-vmx1
                                                vfp_ext-vmx1
    br-int-vmx1     8000.525400b00308   yes br-int-vmx1-nic
                                                vcp_int-vmx1
                                                vfp_int-vmx1
    virbr0      8000.000000000000   yes
```

without login into the VM, VM virtual NIC info (SR-IOV) can be listed with `virsh` command:

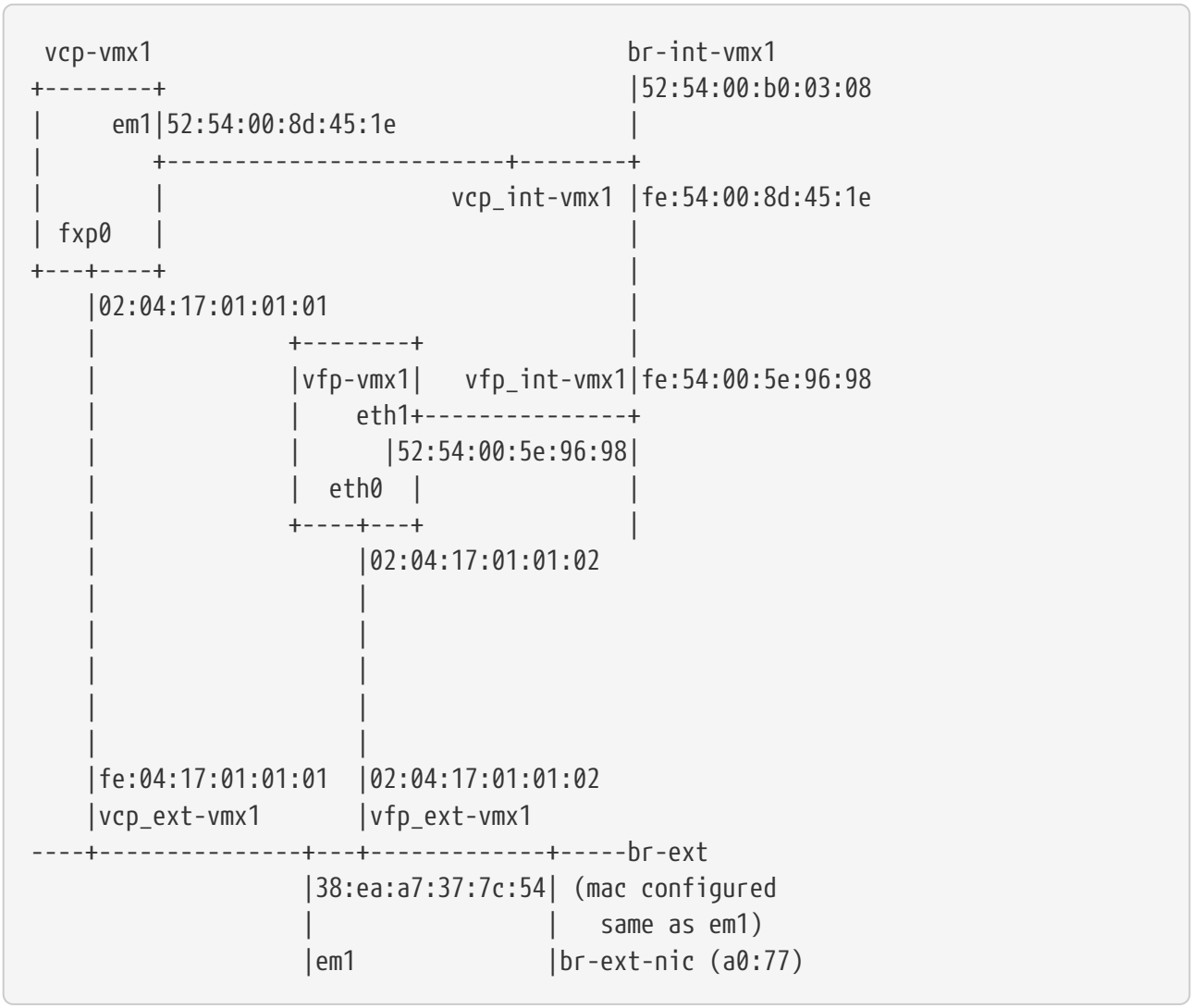
```
    ping@trinity:~$ sudo virsh qemu-monitor-command vcp-vmx1 --hmp "info network"
    net0: index=0,type=nic,model=e1000,macaddr=02:04:17:01:01:01
     \ hostnet0: index=0,type=tap,fd=18
    net1: index=0,type=nic,model=virtio-net-pci,macaddr=52:54:00:8d:45:1e
     \ hostnet1: index=0,type=tap,fd=19
```

```
ping@trinity:~$ sudo virsh qemu-monitor-command vfp-vmx1 --hmp "info network"
net0: index=0,type=nic,model=virtio-net-pci,macaddr=02:04:17:01:01:02
 \ hostnet0: index=0,type=tap,fd=18
net1: index=0,type=nic,model=virtio-net-pci,macaddr=52:54:00:5e:96:98
 \ hostnet1: index=0,type=tap,fd=21
```

11.1. mapping of bridge, tap, guest vNIC (SR-IOV)

this diagram illustrated the relationship between these interfaces:

- external/internal mgmt interfaces in VCP VM: fxp0, em1
- external/internal mgmt interfaces in VFP VM: eth0(ext), eth1(int)
- qemu tap interfaces: vcp(vfp)_ext(int)-vmx1
- linux bridge interface: br-ext, br-int-vmx1



- brige `br-int-vmx1` , with attached qemu tap interfaces vNIC `vcp_int-vmx1` and `vfp_int-vmx1` , and the corresponding guest VM interfaces `em1` and `int`, are for internal communication only, no packets will exit to outside networks, so MAC address does not matters
- brige `br-ext` clones MAC and IP from host mgmt port, so it represents the server from the

external networks' point of view.

- VCP(RE) VM mgmt port `fxp0` and VFP(vFPC) VM mgmt port `ext` or `eth0` will communicate with external network, so they will expose to the outside network and therefore the MAC addresses need to be unique in the segment.
- packets of `fxp0` goes to external network via this path:
  - packet exits `fxp0` of guest VM `vcp-vmx1`
  - packet is received from host tap interface `vcp_ext-vmx1`
  - packet is forwarded to physical mgmt interface `em1` via bridge `br-ext`
  - packet is forwarded to external networks

Table 4. bridge/VN/tap/guest interface mapping table

bridge/VN	forward mode	qemu tap interface	guest VM interface	guest VM
br-int-vmx1	none	`vcp_int-vmx1`	`em1`	VCP(VRE)
		`vfp_int-vmx1`	`int` or `eth1`	VFP(VPFE)
br-ext-vmx1	routed	`vcp_ext-vmx1`	`fxp0`	VCP(VRE)
		`vfp_ext-vmx1`	`ext` or `eth0`	VFP(VPFE)
virbr0	nat			

Chapter 12. vcpu essential

In `vmx.conf` file of previous example, 4 CPUs were allocated to vPFE VM for "lite mode":

```
FORWARDING_PLANE:
    memory-mb    : 16384
    vcpus        : 4
```

In another example we have 16 CPUs allocated for "performance mode":

```
FORWARDING_PLANE:
    memory-mb    : 16384
    vcpus        : 4
```

After logging into vPFE VM we can check the guest VM CPU properties:

vfp

```
root@localhost:~# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                16       #<------
On-line CPU(s) list:   0-15
Thread(s) per core:    1
Core(s) per socket:    16       #<------
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 42
Model name:            Intel Xeon E312xx (Sandy Bridge)
Stepping:              1
CPU MHz:               2992.582
BogoMIPS:              5985.16
Virtualization:        VT-x     #<------
Hypervisor vendor:     KVM      #<------
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              4096K
NUMA node0 CPU(s):     0-15

root@localhost:~# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
```

```
model name      : Intel Xeon E312xx (Sandy Bridge)
stepping        : 1
microcode       : 0x1
cpu MHz         : 2992.582
cache size      : 4096 KB
physical id     : 0
siblings        : 16
core id         : 0
cpu cores       : 16
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
                  pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb
                  rdtscp lm constant_tsc rep_good nopl eagerfpu pni pclmulqdq
                  vmx ssse3 cx16 pcid sse4_1 sse4_2 x2apic popcnt
                  tsc_deadline_timer aes xsave avx f16c rdrand hypervisor
                  lahf_lm xsa veopt vnmi ept fsgsbase smep erms
bogomips        : 5985.16
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:
......
```

The `Hypervisor vendor` indicates the type of hypervisor that the current vCPU was emulated from. In this case it's KVM emulated CPU. In the case of other hypervisor this option will change accordingly. below is an sample taken from a vmware environment:

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    1
Core(s) per socket:    8
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 47
Stepping:              2
CPU MHz:               2394.000
BogoMIPS:              4788.00
Hypervisor vendor:     VMware
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):     0-7
```

for comparison purpose, the original cpu info from the host server is also listed here:

```
ping@ubuntu1:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                20
On-line CPU(s) list:   0-19
Thread(s) per core:    1
Core(s) per socket:    10
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 62
Stepping:              4
CPU MHz:               2992.939
BogoMIPS:              6000.66
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              25600K
NUMA node0 CPU(s):     0-9
NUMA node1 CPU(s):     10-19
```

```
ping@ubuntu1:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 62
model name      : Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz
stepping        : 4
microcode       : 0x427
cpu MHz         : 2992.939
cache size      : 25600 KB
physical id     : 0
siblings        : 10
core id         : 0
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
                  pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
                  syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs
                  bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu
                  pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3
                  cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic popcnt
                  tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm ida arat
                  epb xsaveopt pln pts dtherm tpr_shadow vnmi
                  flexpriority ept vpid fsgsbase smep erms
bogomips        : 5985.87
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

from the comparison it is noticed that the CPU features set (flags) are not identical between the host and guest OS. the host CPU features were provided by the CPU hardware capability and OS support, while the guest CPU features can be either emulated by QEMU software, or simly "exposed" directly from the host CPU capability, by KVM. This was started in libvirt in the form of XML format (see `virsh capabilities`), and passed to `qemu-system-x86_64` process in the form of CPU parameter list (`-cpu`) before the guess VM was brought up.

VMX qemu processes

```
ping@trinity:/virtualization/images/vmx_20151102.0/config$ ps -ef | grep -i qemu
root      17757      1 18 19:01 ?        00:11:45
/usr/bin/qemu-system-x86_64 -name vcp-vmx1 -S -machine
pc-0.13,accel=kvm,usb=off -cpu
SandyBridge,+invtsc,+erms,+smep,+fsgsbase,+pdpe1gb,+rdrand,+f16c,+osxsave,+
dca,+pcid,+pdcm,+xtpr,+tm2,+est,+smx,+vmx,+ds_cpl,+monitor,+dtes64,+pbe,+tm,+ht,+ss,+a
```

```
cpi,+ds,+vme
-m 1954 -realtime mlock=off -smp 1,sockets=1,cores=1,threads=1 -uuid
79ab0bbf-b63a-4a08-87ce-ad7670186754 -smbio s type=0,vendor=Juniper -smbios
type=1,manufacturer=VMX,product=VM-vcp_vmx1-161-re-0,version=0.1.0
-no-user-config -nodefaults -chardev
socket,id=charmonitor,path=//lib/libvirt/qemu/vcp-vmx1.monitor,server,nowai
t -mon chardev=charmonitor,id=monitor,mode=control -rtc base=utc
-no-shutdown -boot strict=on -device
piix3-usb-uhci,id=usb,bus=pci.0,addr=0x1.0x2 -drive
file=/virtualization/images/vmx_20151102.0/build/vmx1/ima
ges/jinstall64-vmx-15.1F-20151104.0-domestic.img,if=none,id=drive-ide0-0-
0,format=qcow2,cache=directsync
-device
ide-hd,bus=ide.0,unit=0,drive=drive-ide0-0-0,id=ide0-0-0,bootindex=1 -drive
file=/virtualization/i
mages/vmx_20151102.0/build/vmx1/images/vmxhdd.img,if=none,id=drive-ide0-0-
1,format=qcow2,cache=directsync
-device ide-hd,bus=ide.0,unit=1,drive=drive-ide0-0-1,id=ide0-0-1 -drive
file=/virtualization/images/vmx_2
0151102.0/images/metadata_usb.img,if=none,id=drive-usb-
disk0,format=raw,cache=directsync
-device usb-storage,drive=drive-usb-disk0,id=usb-disk0,removable=off
-netdev tap,fd=18,id=hostnet0 -device e1000,netdev=ho
stnet0,id=net0,mac=02:04:17:01:01:01,bus=pci.0,addr=0x3 -netdev
tap,fd=19,id=hostnet1,vhost=on,vhostfd=20 -device
virtio-net-pci,netdev=hostnet1,id=net1,mac=52:54:00:84:52:fb,bus=pci.0,addr=0x5
-chardev socket,id=charserial0,host=127.0.0.1,port=8816,telnet,server,nowait -device
isa-serial,chardev=charserial0,id=serial0 -device usb-tablet,id=input0 -vnc
127.0.0.1:0 -device cirrus-vga,id=video0,bus=pci.0,addr=0x2 -device
AC97,id=sound0,bus=pci.0,addr=0x4 -device
virtio-balloon-pci,id=balloon0,bus=pci.0,addr=0x6 -msg timestamp=on

root      17880      1 62 19:01 ?        00:40:31
/usr/bin/qemu-system-x86_64 -name vfp-vmx1 -S -machine
pc-i440fx-trusty,accel=kvm,usb=off,mem-merge=off
-cpu SandyBridge,+invtsc,+erms,+smep,+fsgsbase,+pdpe1gb,+
rdrand,+f16c,+osxsave,+dca,+pcid,+pdcm,+xtpr,+tm2,+est,+smx,+vmx,+ds_cpl,+monitor,+dte
s64,+pbe,+tm,+ht,+ss,+acpi,+ds,+vme
-m 15625 -mem-prealloc -mem-path /HugePage_vPFE/libvirt/qemu -realtime
mlock=off -smp 4,s ockets=1,cores=4,threads=1 -uuid
5d7f31e7-b678-4605-973f-0cb6a3e3b8c1 -no-user-config -nodefaults -chardev
socket,id=charmonitor,path=//lib/libvirt/qemu/vfp-vmx1.monitor,server,nowait
-mon chardev=charmonitor,id =monitor,mode=control -rtc base=utc
-no-shutdown -boot strict=on -device
piix3-usb-uhci,id=usb,bus=pci.0,addr=0x1.0x2 -drive
file=/virtualization/images/vmx_20151102.0/build/vmx1/images/vFPC-20151102.img,if=none
,id=drive-ide0-0-0,format=raw,cache=directsync -device
ide-hd,bus=ide.0,unit=0,drive=drive-ide0-0-0,id=ide0-0-0,bootindex=1
-netdev tap,fd=18,id=hostnet0,vhost=on,vhostfd=20 -device
virtio-net-pci,netdev=hostnet
0,id=net0,mac=02:04:17:01:01:02,bus=pci.0,addr=0x3 -netdev
```

```
tap,fd=21,id=hostnet1,vhost=on,vhostfd=22 -device
virtio-net-pci,netdev=hostnet1,id=net1,mac=52:54:00:c0:ff:1f,bus=pci.0,addr=0x4
-chardev socket,id=charserial0,host=127.0.0.1,port=8817,telnet,server,nowait -device
isa-serial,chardev=charserial0,id=serial0 -device usb-tablet,id=input0 -vnc
127.0.0.1:1 -device cirrus-vga,id=video0,bus=pci.0,addr=0x2 -device AC97
,id=sound0,bus=pci.0,addr=0x5 -device
pci-assign,configfd=23,host=23:10.0,id=hostdev0,bus=pci.0,addr=0x6 -device
pci-assign,configfd=24,host=06:10.0,id=hostdev1,bus=pci.0,addr=0x7 -device
virtio-balloon-pci,id=b alloon0,bus=pci.0,addr=0x8 -msg timestamp=on
```

Here are some highlights about QEMU processes:

- `/usr/bin/qemu-system-x86_64` is the QEMU command to start a VM process
- the long list of QEMU parameters describe how the process will run and what resources will be allocated to it
- the break-down of all these QEMU parameters will be covered later (TODO)
- the VCP(RE) and VFP(FPC) VM are each running as a QEMU process.

  ```
  ping@trinity:~$ ps -ef | grep qemu | cut -c -105
  root      42025      1 18 Nov23 ?        02:40:58 /usr/bin/qemu-system-x86_64 -name
  vcp-vmx1 -S -machine
  root      42148      1 63 Nov23 ?        09:24:41 /usr/bin/qemu-system-x86_64 -name
  vfp-vmx1 -S -machine
  ```

- each VCPU(Virtual CPU) is essentially just a "thread" running inside one of the QEMU processes

threads (vCPU):

```
ping@trinity:~$ sudo virsh qemu-monitor-command 2 --hmp "info cpus"
* CPU #0: pc=0xffffffff80921b83 thread_id=42028             ①

ping@trinity:~$ sudo virsh qemu-monitor-command 3 --hmp "info cpus"
* CPU #0: pc=0xffffffff8100af50 (halted) thread_id=42151    ②
  CPU #1: pc=0xffffffff8100af50 (halted) thread_id=42152    ②
  CPU #2: pc=0xffffffff8100af50 (halted) thread_id=42153    ②
  CPU #3: pc=0xffffffff8100af50 (halted) thread_id=42154    ②

ping@trinity:~$ ps -ef | grep qemu | cut -c -105
root      42025      1 18 Nov23 ?        02:40:58 /usr/bin/qemu-system-x86_64 -name
vcp-vmx1 -S -machine
root      42148      1 63 Nov23 ?        09:24:41 /usr/bin/qemu-system-x86_64 -name
vfp-vmx1 -S -machine

ping@trinity:~$ ps -efL | grep qemu | cut -c -105
root      42025      1  42025 12    4 Nov23 ?        01:47:36 /usr/bin/qemu-system-
x86_64 -name vcp-vmx1
root      42025      1  42028  5    4 Nov23 ?        00:52:06 /usr/bin/qemu-system-
x86_64 -name vcp-vmx1    ①
root      42025      1  42030  0    4 Nov23 ?        00:00:00 /usr/bin/qemu-system-
x86_64 -name vcp-vmx1
root      42025      1  46726  0    4 11:51 ?        00:00:00 /usr/bin/qemu-system-
x86_64 -name vcp-vmx1

root      42148      1  42148  0    7 Nov23 ?        00:00:14 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1
root      42148      1  42151  9    7 Nov23 ?        01:21:37 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1    ②
root      42148      1  42152 25    7 Nov23 ?        03:42:01 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1    ②
root      42148      1  42153 20    7 Nov23 ?        02:58:23 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1    ②
root      42148      1  42154  9    7 Nov23 ?        01:20:30 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1    ②
root      42148      1  42157  0    7 Nov23 ?        00:00:00 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1
root      42148      1  46722  0    7 11:51 ?        00:00:00 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1
```

another capture:

```
ping@trinity:$ sudo virsh qemu-monitor-command 2 --hmp "info cpus"
* CPU #0: pc=0xffffffff80911183 (halted) thread_id=17760

ping@trinity:/virtualization/images/vmx_20151102.0/config$ sudo virsh qemu-monitor-
command 3 --hmp "info cpus"
* CPU #0: pc=0xffffffff8100af50 (halted) thread_id=17883
  CPU #1: pc=0xffffffff8100af50 (halted) thread_id=17884
  CPU #2: pc=0xffffffff8100af50 thread_id=17885
  CPU #3: pc=0xffffffff8100af50 (halted) thread_id=17886

ping@trinity:$ ps -ef | grep qemu | cut -c -105
root      17757      1 17 19:01 ?        00:12:14 /usr/bin/qemu-system-x86_64 -name
vcp-vmx1 -S -machine
root      17880      1 62 19:01 ?        00:42:29 /usr/bin/qemu-system-x86_64 -name
vfp-vmx1 -S -machine

ping@trinity:$ ps -efL | grep qemu | cut -c -105
root      17757      1  17757 11    3 19:01 ?        00:07:58 /usr/bin/qemu-system-
x86_64 -name vcp-vmx1
root      17757      1  17760  6    3 19:01 ?        00:04:08 /usr/bin/qemu-system-
x86_64 -name vcp-vmx1
root      17757      1  17762  0    3 19:01 ?        00:00:00 /usr/bin/qemu-system-
x86_64 -name vcp-vmx1
root      17880      1  17880  0    6 19:01 ?        00:00:11 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1
root      17880      1  17883  9    6 19:01 ?        00:06:37 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1
root      17880      1  17884 24    6 19:01 ?        00:17:01 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1
root      17880      1  17885 19    6 19:01 ?        00:13:29 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1
root      17880      1  17886  7    6 19:01 ?        00:05:14 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1
root      17880      1  17889  0    6 19:01 ?        00:00:00 /usr/bin/qemu-system-
x86_64 -name vfp-vmx1
```

① RE VCPU thread

② PFE VCPU thread

Unix `thread` vs `process`

a *nix Thread, is a "flow of execution" of a process, from the "coding" perspecive it's nothing but (almost) just a "procedure" that runs independently from its main program. Internally from the OS perspective, A thread consists of:

- Program counter
- Register set
- Stack space

So if a main program contains a number of procedures. all of these procedures are able to be scheduled to run simultaneously and/or independently by the operating system.

A thread is running inside of a process, and differs with the concept of a "process" in the sense that thread shares with its peer threads its:

- Code segment
- Data segment
- Operating-system resources

e.g., when one thread alters a code segment memory item, all other threads see that, and a file open with one thread is available to others. this is different with a "process" in that each process has its own running space and resources. communicating between processes (called "IPC") is more "expensive" and requires some special OS utilities.

In the context of VMX, "worker thread", and "IO thread" are examples of threads running in the host OS.

Chapter 13. memory

memory allocated in `vmx.conf` is 16G:

```
#vPFE VM parameters
FORWARDING_PLANE:
    memory-mb   : 16384
    vcpus       : 4                    ①
```

memory available in guest VM:

vPFE VM:

```
root@localhost:~# free -m
             total       used       free     shared    buffers     cached
Mem:         15297      11380       3917          0         13       2697
-/+ buffers/cache:       8669       6628
Swap:            0          0          0
```

this looks a little bit smaller than expected: `16384-15297=1087M`.

So about 1G memory was "missing".

By looking at kernel boot log we will find more clues about what really happened:

```
root@localhost:~# dmesg -T | grep -i memory
[Sun Dec 20 23:59:58 2015] Scanning 1 areas for low memory corruption
[Sun Dec 20 23:59:58 2015] Base memory trampoline at [ffff880000099000] 99000 size
24576
[Sun Dec 20 23:59:58 2015] init_memory_mapping: [mem 0x00000000-0x000fffff]
[Sun Dec 20 23:59:58 2015] init_memory_mapping: [mem 0x410600000-0x4107fffff]
[Sun Dec 20 23:59:58 2015] init_memory_mapping: [mem 0x410000000-0x4105fffff]
[Sun Dec 20 23:59:58 2015] init_memory_mapping: [mem 0x400000000-0x40fffffff]
[Sun Dec 20 23:59:58 2015] init_memory_mapping: [mem 0x00100000-0xbfffbfff]
[Sun Dec 20 23:59:58 2015] init_memory_mapping: [mem 0x100000000-0x3fffffff]
[Sun Dec 20 23:59:58 2015] init_memory_mapping: [mem 0x410800000-0x4108fffff]
[Sun Dec 20 23:59:58 2015] Early memory node ranges
[Sun Dec 20 23:59:58 2015] Memory: 15660932k/<strong>17048576k</strong> available
(8700k kernel code, 1048984k absent, 338660k reserved, 6548k data, 1172k init)
[Sun Dec 20 23:59:58 2015] please try 'cgroup_disable=memory' option if you don't want
memory cgroups
[Sun Dec 20 23:59:58 2015] Initializing cgroup subsys memory
[Sun Dec 20 23:59:58 2015] Freeing initrd memory: 728k freed
[Sun Dec 20 23:59:58 2015] Scanning for low memory corruption every 60 seconds
[Sun Dec 20 23:59:59 2015] Freeing unused kernel memory: 1172k freed
[Sun Dec 20 23:59:59 2015] Freeing unused kernel memory: 1528k freed
[Sun Dec 20 23:59:59 2015] Freeing unused kernel memory: 584k freed
```

So Kernel detected `17048576k` (`=16649M`), which is much closer with what was allocated. The rest of memory is what kernel reserved for internal tasks.

`/proc/meminfo` reveals more details about memory usage.

```
root@localhost:~# cat /proc/meminfo
MemTotal:       15664972 kB
MemFree:         4010928 kB
Buffers:           13956 kB
Cached:          2762276 kB
SwapCached:            0 kB
Active:           148364 kB
Inactive:        2723568 kB
Active(anon):      86864 kB
Inactive(anon):  2665792 kB
Active(file):      61500 kB
Inactive(file):    57776 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:        112212 kB
Mapped:           177764 kB
Shmem:           2640572 kB
Slab:              70948 kB
SReclaimable:      30708 kB
SUnreclaim:        40240 kB
KernelStack:        1968 kB
PageTables:         2712 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     3638180 kB
Committed_AS:    3430712 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       51268 kB
VmallocChunk:   34359683660 kB
HugePages_Total:    4096
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       13296 kB
DirectMap2M:     2355200 kB
DirectMap1G:    13631488 kB
```

Chapter 14. virtio bridging

virtio provides emulations of NIC , disk and other IO devices with high performance, but it does not has any "build-in" bridging facilities or anything in that purpose. So in order to "bind" the virtio interfaces to the target physical NIC, we usually need to add that "bridging" into our virtio VMX setup after the VMs were brought up and running. This can be done with one of the existing utilities that is available, depending on whichever is suitable for the needs.

some popular examples of these utilities will be:

- legacy linux `bridge`,
- open `vSwitch` (OVS),
- Juniper `vRouter`.

Here is a simplest example of "binding" 2 virtio interfaces to each other, with linux bridge.

before "binding"

```
ping@trinity:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
br-ext          8000.38eaa7377c54       yes             br-ext-nic
                                                        em1
                                                        vcp_ext-vmx1
                                                        vfp_ext-vmx1
br-int-vmx1             8000.5254006b156e       yes     br-int-vmx1-nic
                                                        vcp_int-vmx1
                                                        vfp_int-vmx1
virbr0          8000.fe0417010201       yes             ge-0.0.0-vmx1
                                                        ge-0.0.1-vmx1
```

"binding" 2 virtio interfaces is like to connect a "loopback cable" between them , say , `ge-0/0/0` and `ge-0/0/1`. Changing the `config/vmx-junosdev.conf` file so it looks:

```
ping@trinity:/virtualization/images/vmx_20151102.0/config$ vim vmx-junosdev.conf.1
################################################################
#
#  vmx-junos-dev.conf
#  - Config file for junos device bindings.
#  - Uses YAML syntax.
#  - Leave a space after ":" to specify the parameter value.
#  - For physical NIC, set the 'type' as 'host_dev'
#  - For junos devices, set the 'type' as 'junos_dev' and
#    set the mandatory parameter 'vm-name' to the name of
#    the vPFE where the device exists
#  - For bridge devices, set the 'type' as 'bridge_dev'
#
################################################################
interfaces :

     - link_name  : vmx_link4
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx1
           dev_name    : ge-0/0/0
       endpoint_2 :
         - type        : junos_dev
           vm_name     : vmx1
           dev_name    : ge-0/0/1
```

This read as: "please create a linux bridge `vmx_link4`, and use it to bind junos device interface `ge-0/0/0` of VM vmx1, to the other juno device interface `ge-0/0/1` in the same VM"

Now run the vmx.sh script the execute this action:

```
sudo ./vmx.sh -lvf --bind-dev --cfg config/vmx-junosdev.conf.1
Checking package ethtool.........................[OK]
Bind Link vmx_link4(ge-0.0.0-vmx1, ge-0.0.1-vmx1)
[OK]
```

As expected, this will create a new bridge named `vmx_link4` and add the JUNOS interfaces (`ge-0/0/0` and `ge-0/0/1`) mapped corresponding virtio tap interface (`ge-0.0.0-vmx1` and `ge-0.0.1-vmx1`) into the bridge.

```
ping@trinity:/virtualization/images/vmx_20151102.0$ brctl show
bridge name     bridge id               STP enabled     interfaces
br-ext          8000.38eaa7377c54       yes             br-ext-nic
                                                        em1
                                                        vcp_ext-vmx1
                                                        vfp_ext-vmx1

br-int-vmx1             8000.525400fa6e56       yes     br-int-vmx1-nic
                                                        vcp_int-vmx1
                                                        vfp_int-vmx1

virbr0          8000.000000000000       yes                             ①

vmx_link4               8000.fe0417010201       no      ge-0.0.0-vmx1   ②
                                                        ge-0.0.1-vmx1
```

① the links were removed from the default `virbr0` bridge

② the links were added to new bridge `vmx_link4`.

14.1. build a setup with internal connections

now we can setup a Junos logical router based virtual test environment like below:

```
192.168.122.10            192.168.122.3
        .....vmx_link4........
        .                    .
        .                    .
   +----.--------------------.-----+
   | ge-0.0.0-vmx1  ge-0.0.1-vmx1  |
   |    .                    .     |
   |    .                    .     |
   | +--.-------+    +-------.--+  |
   | |(ge-0/0/0)|    |(ge-0/0/1)|  |
   | |192.168.  |    |192.168.  |  |
   | |122.10    |    |122.3     |  |
   | |          |    |          |  |
   | |LR:default|    |LR: r1    |  |
   | +----------+    +----------+  |
   |                               |
   +-------------------------------+
```

with logical routers (LR) and internal connection between the virtio interfaces, we now have a handy virtual multi-router setup.

ping test:

```
[edit]
root# run show interfaces routing
Interface        State Addresses
ge-0/0/0.0       Up    INET  192.168.122.10         #<------
pfe-0/0/0.16383  Up
pfh-0/0/0.16384  Up
pfh-0/0/0.16383  Up
lc-0/0/0.32769   Up
lo0.16385        Up
lo0.16384        Up    INET  127.0.0.1
fxp0.0           Up    INET  192.168.1.105
em1.0            Up    INET  10.0.0.4
                       INET  128.0.0.1
                       INET  128.0.0.4
                       INET6 fe80::5254:ff:fe5b:901e
                       INET6 fec0::a:0:0:4

[edit]
root# run show interfaces routing logical-system r1
Interface        State Addresses
ge-0/0/1.0       Up    INET  192.168.122.3          #<------

[edit]
root# run ping 192.168.122.3
PING 192.168.122.3 (192.168.122.3): 56 data bytes
64 bytes from 192.168.122.3: icmp_seq=0 ttl=63 time=15.073 ms
^C
--- 192.168.122.3 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 15.073/15.073/15.073/nan ms

[edit]
root# run ping 192.168.122.10 logical-system r1
PING 192.168.122.10 (192.168.122.10): 56 data bytes
64 bytes from 192.168.122.10: icmp_seq=0 ttl=64 time=2.058 ms
^C
--- 192.168.122.10 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.058/2.058/2.058/0.000 ms
```

Part 3: VMX/KVM features

in this part I'll explore some of the common, and key VMX-involved KVM features. For each topic, I'll start with the "basic concept", then proceed with command illustrations.

Chapter 15. VM management - virsh

15.1. virsh basic concept

`Libvirt` is collection of open source API, daemon and management tool that provides a convenient way to manage virtual machines and other virtualization functionality, such as storage and network interface management. The software components include:

- an API library,
- a daemon (libvirtd), and
- a command line utility (virsh).

It can be used to manage KVM, Xen, VMware ESX, QEMU and a lot of other (may all?) currently popular virtualization technologies.

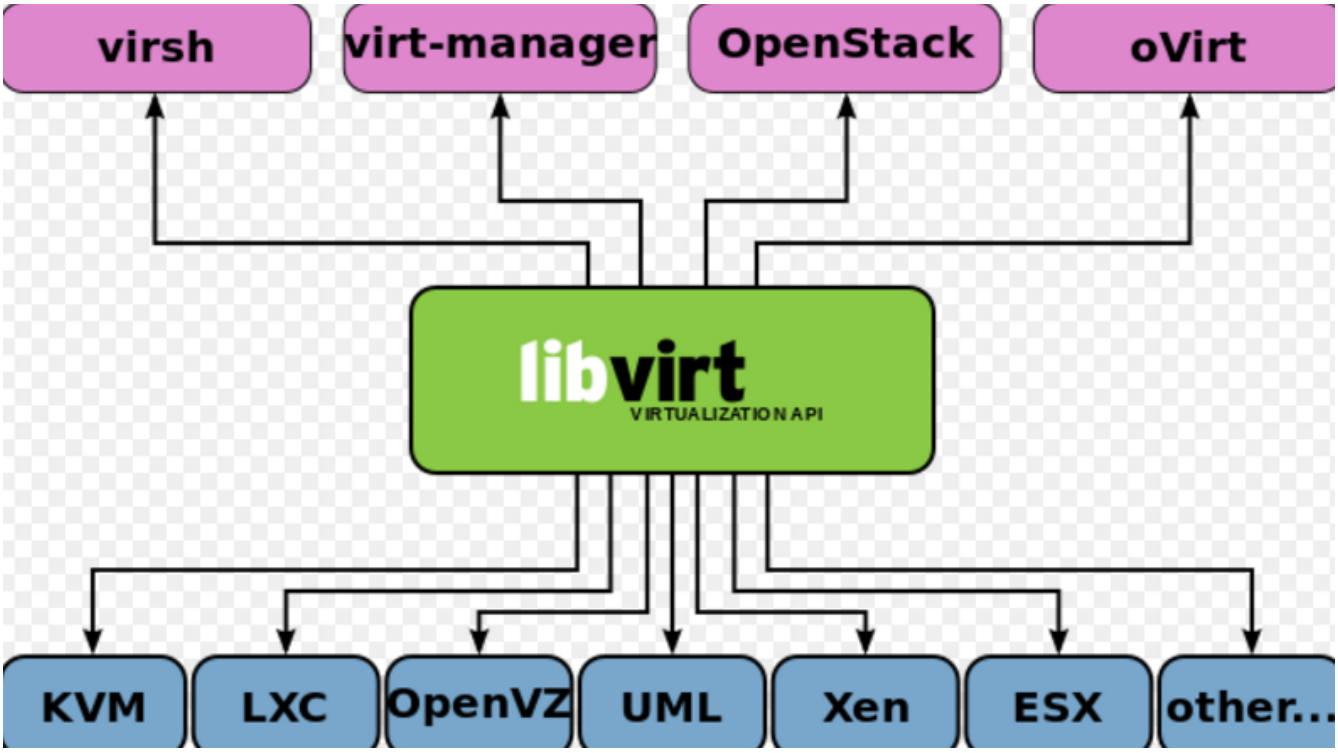


Figure 5. libvirt

A primary goal of libvirt is to provide a single (and simple) way to manage multiple different virtualization providers/hypervisors - same command can be used to manage the existing virtual machines for any supported hypervisor (KVM, Xen, VMWare ESX, etc.)

For example: To list current running guest VMs:

```
ping@trinity:~$ sudo virsh list
[sudo] password for ping:
 Id    Name                           State
----------------------------------------------------
 2     vcp-vmx1                       running
 3     vfp-vmx1                       running
```

The above command will list any running VMs that are managed by `libvirt`, regardless of which hypervisor is currently in use. In this doc we'll focus on VMX, which is currently implemented based on `qemu-kvm` hypervisor.

use `--all` option to list all guest VMs including those not currently running.

`Libvirt` also provides extensive tools and commands to manage the `domain`, `node` and `network`.

In `libvirt` terminology, `domain` indicates guest VM and `node` refers to the host machine.

15.2. libvirt/virsh commonly used commands

In this section we'll demonstrate some of most commonly used virsh commands to manage the host and guest OS. refer to virsh online help (`virsh help`) or libvirt website for more details of the usage.

15.2.1. domain managment

dominfo: (virtual) CPU/Memory info of each domain(VM):

```
ping@trinity:~$ sudo virsh dominfo 2
Id:             2
Name:           vcp-vmx1
UUID:           90520384-41af-4029-9523-040ec59bb7f2
OS Type:        hvm
State:          running
CPU(s):         1       #<------
CPU time:       7461.9s
Max memory:     2000896 KiB     #<------
Used memory:    2000000 KiB     #<------
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: none
Security DOI:   0

ping@trinity:~$ sudo virsh dominfo 3
Id:             3
Name:           vfp-vmx1
UUID:           34991bba-ecbf-4680-905f-dc1d6b1c2308
OS Type:        hvm
State:          running
CPU(s):         4       #<------
CPU time:       29660.4s
Max memory:     16000000 KiB    #<------
Used memory:    16000000 KiB    #<------
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: none
Security DOI:   0
```

vcpuinfo: state of each virtual cpu assigned to guest VM

```
ping@trinity:~$ sudo virsh vcpuinfo vcp-vmx1
VCPU:           0
CPU:            7
State:          running
CPU time:       465.7s
CPU Affinity:   -------y------------------------

ping@trinity:~$ sudo virsh vcpuinfo vfp-vmx1
VCPU:           0
CPU:            0
State:          running
CPU time:       571.7s
CPU Affinity:   y------------------------------

VCPU:           1
CPU:            1
State:          running
CPU time:       1432.5s
CPU Affinity:   -y-----------------------------

VCPU:           2
CPU:            2
State:          running
CPU time:       1133.6s
CPU Affinity:   --y----------------------------

VCPU:           3
CPU:            3
State:          running
CPU time:       517.1s
CPU Affinity:   ---y---------------------------
```

the "CPU affinity" indicates the "binding" between vcpu in guest VM and physical cpu core in host machine [5: or, it can be "logical" cpu core when hyperthreading is enabled]. This can be archived by using `vcpupin` command below.

vcpupin: display or control the vcpu to host cpu affinity

```
virsh emulatorpin vcp-vmx1 0
virsh emulatorpin vfp-vmx1 0

virsh vcpupin vcp-vmx1 0 7

virsh vcpupin vfp-vmx1 0 0
virsh vcpupin vfp-vmx1 1 1
virsh vcpupin vfp-vmx1 2 2
virsh vcpupin vfp-vmx1 3 3
```

With these commands, we can bind VCPU 0 of `vcp-vmx1` VM to core 7 of host machine, VCPU 0 of `vfp-vmx1` VM to core 0 of host machine, so on so forth. in the section of VCPU essential, we know that each "VCPU" is essentially just a thread running in the space of one of the two QEMU processes, so binding a "VCPU" to a host cpu core is essentially just to "scope" the running of a thread to a specific CPU core in the host - that is why this feature is called "CPU affinity".

The `emulatorpin` indicate the thread of hypervisor itself.

vcpucount: count of VCPU in use by a VM

```
ping@ubuntu1404:~$ sudo virsh vcpucount contrail
maximum      config         2
maximum      live           2
current      config         2
current      live           2
```

storage/images of each VM:

```
    ping@trinity:~$ sudo virsh domblklist 2
    Target     Source
    ------------------------------------------------
    hda        /virtualization/images/vmx_20151102.0/build/vmx1/images/jinstall64-vmx-
15.1F-20151104.0-domestic.img
    hdb        /virtualization/images/vmx_20151102.0/build/vmx1/images/vmxhdd.img
    sda        /virtualization/images/vmx_20151102.0/images/metadata_usb.img
```

```
ping@trinity:~$ sudo virsh domblklist 3
Target     Source
------------------------------------------------
hda        /virtualization/images/vmx_20151102.0/build/vmx1/images/vFPC-20151102.img
```

This command is pretty handy that with it we can quickly locate from which images the current VMs were built from, and where are the image files currently located.

useful qemu-monitor-command: this can be used to print the qmp command

```
ping@trinity:~$ sudo virsh qemu-monitor-command vcp-vmx1 --hmp "info network"
net0: index=0,type=nic,model=e1000,macaddr=02:04:17:01:01:01
 \ hostnet0: index=0,type=tap,fd=18
net1: index=0,type=nic,model=virtio-net-pci,macaddr=52:54:00:8d:45:1e
 \ hostnet1: index=0,type=tap,fd=19

ping@trinity:~$ sudo virsh qemu-monitor-command vfp-vmx1 --hmp "info network"
net0: index=0,type=nic,model=virtio-net-pci,macaddr=02:04:17:01:01:02
 \ hostnet0: index=0,type=tap,fd=18
net1: index=0,type=nic,model=virtio-net-pci,macaddr=52:54:00:5e:96:98
 \ hostnet1: index=0,type=tap,fd=21

ping@trinity:~$ sudo virsh qemu-monitor-command 2 --hmp "info cpus"
* CPU #0: pc=0xffffffff80921b83 thread_id=42028

ping@trinity:~$ sudo virsh qemu-monitor-command 3 --hmp "info cpus"
* CPU #0: pc=0xffffffff8100af50 (halted) thread_id=42151
  CPU #1: pc=0xffffffff8100af50 (halted) thread_id=42152
  CPU #2: pc=0xffffffff8100af50 (halted) thread_id=42153
  CPU #3: pc=0xffffffff8100af50 (halted) thread_id=42154
```

This is sometimes quite useful because it removes the need to acquire a qmp console of a guest VM in order to send qmp command.

The QEMU Machine Protocol (QMP) is a JSON-based protocol which allows applications to control a QEMU instance. in another word you can query or manipulate the running status of a VM ,without even "login" into the VM itself.

15.2.2. node management

"node" means host machine under the context of virsh. So all commands listed here is about the host, not the VM - This effectively provides another unified CLI to manage the host machine, regardless of the OS types. Considering there are so many OS variations and each may come with different , aften incompatible CLIes, using virsh may be easier to identify some common data about the host.

nodeinfo: cpu+memory brief overview

```
ping@trinity:~$ sudo virsh nodeinfo
CPU model:           x86_64
CPU(s):              32
CPU frequency:       3300 MHz
CPU socket(s):       1
Core(s) per socket:  8
Thread(s) per core:  1
NUMA cell(s):        4
Memory size:         528417400 KiB

labroot@MX86-host-BL660C-B1:~$ sudo virsh nodeinfo
CPU model:           x86_64
CPU(s):              32
CPU frequency:       1200 MHz
CPU socket(s):       1
Core(s) per socket:  8
Thread(s) per core:  1
NUMA cell(s):        4
Memory size:         528417400 KiB
```

sysinfo: host system general info in more detail (bios/cpu/memory/etc)

```
ping@trinity:~$ sudo virsh sysinfo
<sysinfo type='smbios'>
  <bios>        #<------similar to "dmidecode -s bios-version"
    <entry name='vendor'>HP</entry>
    <entry name='version'>I32</entry>
    <entry name='date'>02/10/2014</entry>
  </bios>
  <system>      #<------similar to "dmidecode -t 1"
    <entry name='manufacturer'>HP</entry>
    <entry name='product'>ProLiant BL660c Gen8</entry>
    <entry name='version'>Not Specified</entry>
    <entry name='serial'>USE4379WSS      </entry>
    <entry name='uuid'>31393736-3831-5355-4534-333739575353</entry>
    <entry name='sku'>679118-B21      </entry>
    <entry name='family'>ProLiant</entry>
  </system>
  <processor>   #<------CPU socket: similar to "dmidecode -t 4"
    <entry name='socket_destination'>Proc 1</entry>
    <entry name='type'>Central Processor</entry>
    <entry name='family'>Xeon</entry>
    <entry name='manufacturer'>Intel</entry>
    <entry name='signature'>Type 0, Family 6, Model 62, Stepping 4</entry>
    <entry name='version'> Intel(R) Xeon(R) CPU E5-4627 v2 @ 3.30GHz</entry>
    <entry name='external_clock'>100 MHz</entry>
    <entry name='max_speed'>4800 MHz</entry>
    <entry name='status'>Populated, Enabled</entry>
```

```
    <entry name='serial_number'>Not Specified</entry>
    <entry name='part_number'>Not Specified</entry>
  </processor>
  <processor>
    ......
  </processor>
  <memory_device> #<------similar to "dmidecode -t 17"
    <entry name='size'>32 GB</entry>
    <entry name='form_factor'>DIMM</entry>
    <entry name='locator'>PROC  1 DIMM  1</entry>
    <entry name='bank_locator'>Not Specified</entry>
    <entry name='type'>DDR3</entry>
    <entry name='type_detail'>Synchronous</entry>
    <entry name='speed'>1866 MHz</entry>
    <entry name='manufacturer'>HP</entry>
    <entry name='serial_number'>Not Specified</entry>
    <entry name='part_number'>712384-081</entry>
  </memory_device>
  <memory_device>
  ....
  </memory_device>
  ....

</sysinfo>
```

freecell: available memory

```
ping@trinity:~$ sudo virsh freecell
Total: 520897088 KiB
```

nodememstats

```
ping@ubuntu1404:~$ sudo virsh nodememstats
total  :              8033536 KiB
free   :              2874116 KiB
buffers:              1333740 KiB
cached :              1462096 KiB
```

capabilities: system supported features

```
ping@trinity:~$ sudo virsh capabilities
<capabilities>

  <host>
    <uuid>36373931-3138-5553-4534-333739575353</uuid>
    <cpu>       #<------cpu flags: "cat /proc/cpuinfo"
      <arch>x86_64</arch>
      <model>SandyBridge</model>
      <vendor>Intel</vendor>
```

```
      <topology sockets='1' cores='8' threads='1'/>
      <feature name='invtsc'/>
      <feature name='erms'/>
      <feature name='smep'/>
      <feature name='fsgsbase'/>
      <feature name='pdpe1gb'/>
      <feature name='rdrand'/>
      <feature name='f16c'/>
      <feature name='osxsave'/>
      <feature name='dca'/>
      <feature name='pcid'/>
      <feature name='pdcm'/>
      <feature name='xtpr'/>
      <feature name='tm2'/>
      <feature name='est'/>
      <feature name='smx'/>
      <feature name='vmx'/>
      <feature name='ds_cpl'/>
      <feature name='monitor'/>
      <feature name='dtes64'/>
      <feature name='pbe'/>
      <feature name='tm'/>
      <feature name='ht'/>
      <feature name='ss'/>
      <feature name='acpi'/>
      <feature name='ds'/>
      <feature name='vme'/>
      <pages unit='KiB' size='4'/>
      <pages unit='KiB' size='2048'/>
    </cpu>
    <power_management>
      <suspend_disk/>
      <suspend_hybrid/>
    </power_management>
    <migration_features>
      <live/>
      <uri_transports>
        <uri_transport>tcp</uri_transport>
      </uri_transports>
    </migration_features>
    <topology>
      <cells num='4'>
        <cell id='0'>   #<------NUMA node0
          <memory unit='KiB'>132067432</memory>
          <pages unit='KiB' size='4'>33016858</pages>
          <pages unit='KiB' size='2048'>0</pages>
          <distances>
            <sibling id='0' value='10'/>
            <sibling id='1' value='21'/>
            <sibling id='2' value='21'/>
            <sibling id='3' value='21'/>
```

```
        </distances>
        <cpus num='8'>
          <cpu id='0' socket_id='0' core_id='0' siblings='0'/>
          <cpu id='1' socket_id='0' core_id='1' siblings='1'/>
          <cpu id='2' socket_id='0' core_id='2' siblings='2'/>
          <cpu id='3' socket_id='0' core_id='3' siblings='3'/>
          <cpu id='4' socket_id='0' core_id='4' siblings='4'/>
          <cpu id='5' socket_id='0' core_id='5' siblings='5'/>
          <cpu id='6' socket_id='0' core_id='6' siblings='6'/>
          <cpu id='7' socket_id='0' core_id='7' siblings='7'/>
        </cpus>
      </cell>
      ......
  </topology>
  <secmodel>
    <model>none</model>
    <doi>0</doi>
  </secmodel>
  <secmodel>
    <model>dac</model>
    <doi>0</doi>
    <baselabel type='kvm'>+0:+0</baselabel>
    <baselabel type='qemu'>+0:+0</baselabel>
  </secmodel>
</host>

<guest>
  <os_type>hvm</os_type>
  <arch name='i686'>
    <wordsize>32</wordsize>
    <emulator>/usr/bin/qemu-system-i386</emulator>
    <machine canonical='pc-i440fx-trusty' maxCpus='255'>pc</machine>
    <machine maxCpus='255'>pc-0.12</machine>
      ......
    <machine maxCpus='255'>pc-i440fx-2.0</machine>
    <machine maxCpus='255'>pc-0.13</machine>
    <domain type='qemu'>
    </domain>
    <domain type='kvm'>
      <emulator>/usr/bin/kvm</emulator>
      <machine canonical='pc-i440fx-trusty' maxCpus='255'>pc</machine>
      <machine maxCpus='255'>pc-1.3</machine>
      ......
      <machine maxCpus='255'>pc-0.13</machine>
    </domain>
  </arch>
  <features>
    <cpuselection/>
    <deviceboot/>
    <disksnapshot default='on' toggle='no'/>
    <acpi default='on' toggle='yes'/>
```

```
      <apic default='on' toggle='no'/>
      <pae/>
      <nonpae/>
    </features>
  </guest>

  <guest>
    ......
  </guest>

</capabilities>
```

this output is quite long so not everything is printed here. for completeness the whole capture is available in appendix

15.2.3. network and interface management

network and interface management virsh commands are used to manage the network resources in the host.

net-list: list active (or all if also with `--all`) virtual networks

```
labroot@MX86-host-BL660C-B1:~$ sudo virsh net-list --all
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 br-ext               active     no            yes
 br-int               active     no            yes
 default              active     yes           yes
```

net-info: print properties of a specific network

```
labroot@MX86-host-BL660C-B1:~$ sudo virsh net-info br-ext
Name:           br-ext
UUID:           efef3d11-cf4f-4cc2-9d21-d1814c0e2e0e
Active:         yes
Persistent:     yes
Autostart:      no
Bridge:         br-ext
```

```
labroot@MX86-host-BL660C-B1:~$ sudo virsh net-info br-int
Name:           br-int
UUID:           e8af7e46-f574-4d8d-96ef-210a8d61067b
Active:         yes
Persistent:     yes
Autostart:      no
Bridge:         br-int
```

```
labroot@MX86-host-BL660C-B1:~$ sudo virsh net-info default
Name:           default
UUID:           0969babb-7b8f-457d-8548-1783152ad05d
Active:         yes
Persistent:     yes
Autostart:      yes
Bridge:         virbr0
```

iface-list: list all active (up) interfaces in host

```
ping@trinity:~$ sudo virsh iface-list
 Name                 State      MAC Address
---------------------------------------------------
 br-ext               active     38:ea:a7:37:7c:54
 p2p1                 active     38:ea:a7:17:65:a0
 p3p1                 active     38:ea:a7:17:65:84
```

```
ping@matrix:/home$ sudo virsh iface-list
[sudo] password for ping:
 Name                 State      MAC Address
---------------------------------------------------
 br-int-vmx1          active     52:54:00:04:5f:fe
 br-int-vmx2          active     52:54:00:d8:7b:dc
 br0                  active     5c:b9:01:8a:f0:b8
 br1                  active     5c:b9:01:8a:f0:b9
 p1p1                 active     8c:dc:d4:b7:79:d0
 p1p2                 active     8c:dc:d4:b7:79:d1
 p2p1                 active     8c:dc:d4:b7:7a:e8
 p2p2                 active     8c:dc:d4:b7:7a:e9
```

iface-mac: print MAC address of one specific interface in host

```
ping@trinity:~$ sudo virsh iface-mac p2p1
38:ea:a7:17:65:a0
```

Chapter 16. iommu/VT-d

16.1. VT-d basic concept

I/O Virtualization (IOV) involves **sharing** a single I/O resource between multiple virtual machines. Approaches for IOV include (but may not be limited to):

- software based approach
- direct assignment
- SR-IOV

16.1.1. software based approach

Software based sharing utilizes emulation techniques to provide a logical I/O hardware device to the VM. The emulation layer interposes itself between the driver running in the guest OS and the underlying hardware. This level of indirection allows the VMM to intercept all traffic issued by the guest driver.

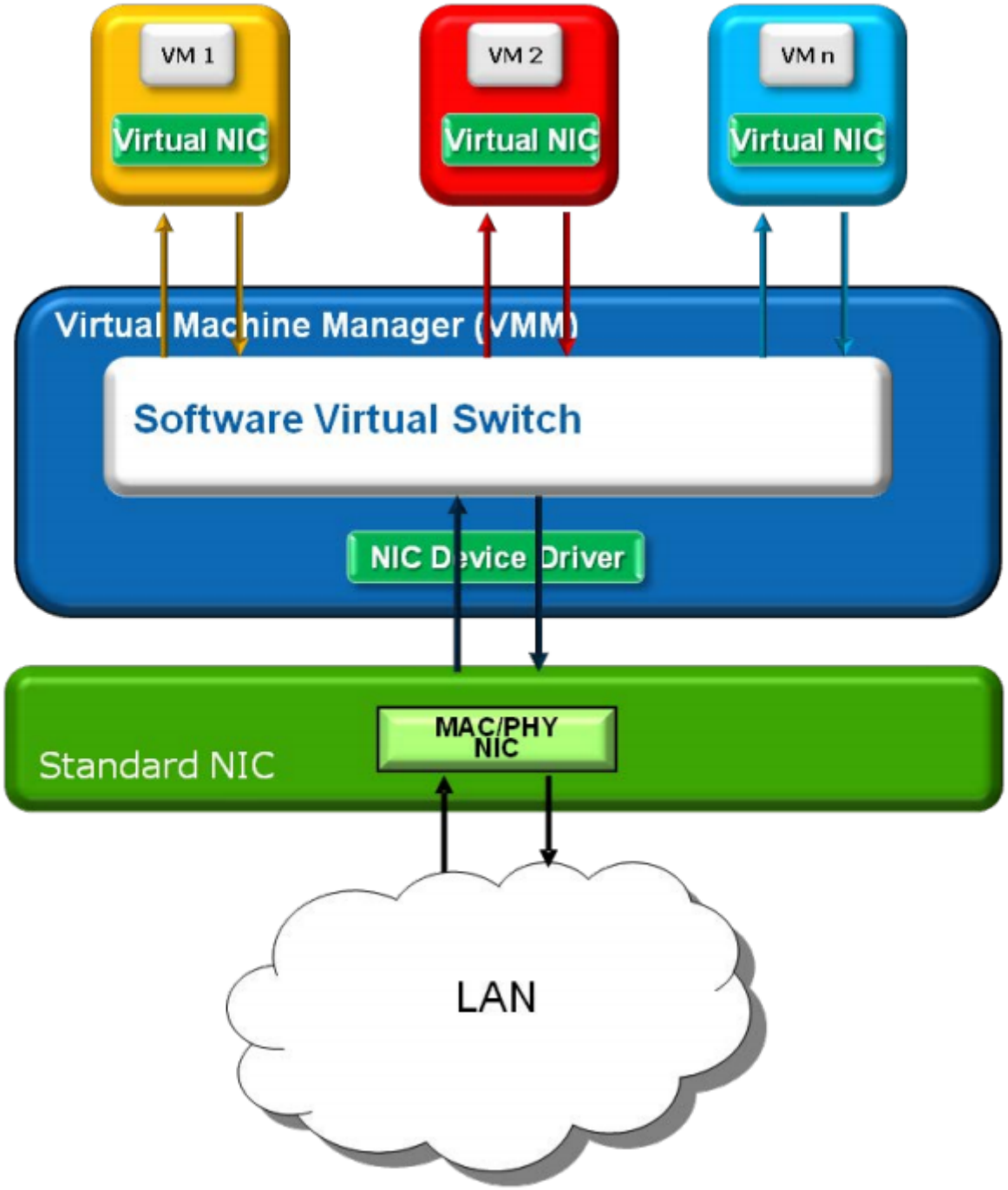


Figure 6. software based I/O sharing

Emulation software involves following common tasks:

- parse the I/O commands,
- translate guest addresses into host physical addresses
- ensure that all referenced memory pages are present in memory.
- resolve the multiple I/O requests from all the virtual machines
- serialize them into a single I/O stream that can be handled by the underlying hardware.

There are at least 2 common approaches to software-based sharing:

- device emulation

- the splitdriver model:

Device emulation models

In this method, the hypervisor mimic widely supported real devices (such as an Intel 1Gb NIC) and utilize existing drivers in the guest OS. The VMM emulates the I/O device to ensure compatibility and then processes I/O operations before passing them on to the physical device (which may be different).

an example of the implementation of this model is QEMU, which can emulate a lot of legacy devices that can be used in the VM, regardless of real physical device type available in the host.

```
ping@trinity:~$ qemu-system-x86_64 -device ?
...<snippet>...
name "ioh3420", bus PCI, desc "Intel IOH device id 3420 PCIE Root Port"
...<snippet>...
name "piix3-ide", bus PCI
name "piix3-ide-xen", bus PCI
...<snippet>...
name "e1000", bus PCI, desc "Intel Gigabit Ethernet"
name "i82550", bus PCI, desc "Intel i82550 Ethernet"
name "i82551", bus PCI, desc "Intel i82551 Ethernet"
name "i82557a", bus PCI, desc "Intel i82557A Ethernet"
name "i82557b", bus PCI, desc "Intel i82557B Ethernet"
name "i82557c", bus PCI, desc "Intel i82557C Ethernet"
name "i82558a", bus PCI, desc "Intel i82558A Ethernet"
...<snippet>...
name "isa-parallel", bus ISA
name "isa-serial", bus ISA
...<snippet>...
name "isa-vga", bus ISA
...<snippet>...
name "adlib", bus ISA, desc "Yamaha YM3812 (OPL2)"
```

A potential problem is that I/O operations then have to traverse two I/O stacks, one in the VM and one in the VMM. Because of the low performance archived, this method is rarely used in an environment where performance is key.

split-driver model

This method takes a similar approach but, instead of emulating a legacy device, the split-driver uses a front-end driver in the guest that works in concert with a back-end driver in the VMM. These drivers are optimized for sharing and have the benefit of not needing to emulate an entire device. The back-end driver communicates with the physical device.

An example of the implementation of this model is `virtio`, which will be discussed in section virtio.

Both the device emulation and split-driver (para-virtualized driver) provide a subset of the total functionality provided by physical hardware and may as a result not have the ability to take advantage of advanced capabilities provided by the device.

In addition, significant CPU overhead may be required by the VMM to implement a virtual software-based switch that routes packets to and from the appropriate VMS. This CPU overhead can (and generally does) reduce the maximum throughput on an I/O device. As an example, extensive testing has shown using only device emulation, a 10Gbps Ethernet controller can achieve a maximum throughput of 4.5 to around 6.5 Gbps (the range varies with the architecture of the server being tested on).

One reason that line rate, or near line rate cannot be achieved is because each packet must go through the software switch and that requires CPU cycles to process the packets.

16.1.2. direct assignment

Software-based sharing adds overhead to each I/O operation due to the emulation layer between the guest driver and the I/O hardware. This indirection has the additional affect of eliminating the use of hardware acceleration that may be available in the physical device.

Such problems can be reduced by directly exposing the hardware to the guest OS and running a native device driver.

Intel has added enhancements to facilitate memory translation and ensure protection of memory that enables a device to directly DMA to/from host memory. These enhancements provide the ability to bypass the VMM's I/O emulation layer and can result in throughput improvement for the VMs.

- Intel® `VT-x` : allows a VM to have direct access to a physical address (if so configured by the VMM). This ability allows a device driver within a VM to be able to write directly to registers IO device (such as configuring DMA descriptors).
- Intel® `VT-d` : provides a similar capability for IO devices to be able to write directly to the memory space of a virtual machine, for example a DMA operation. VT-d technology is illustrated below:

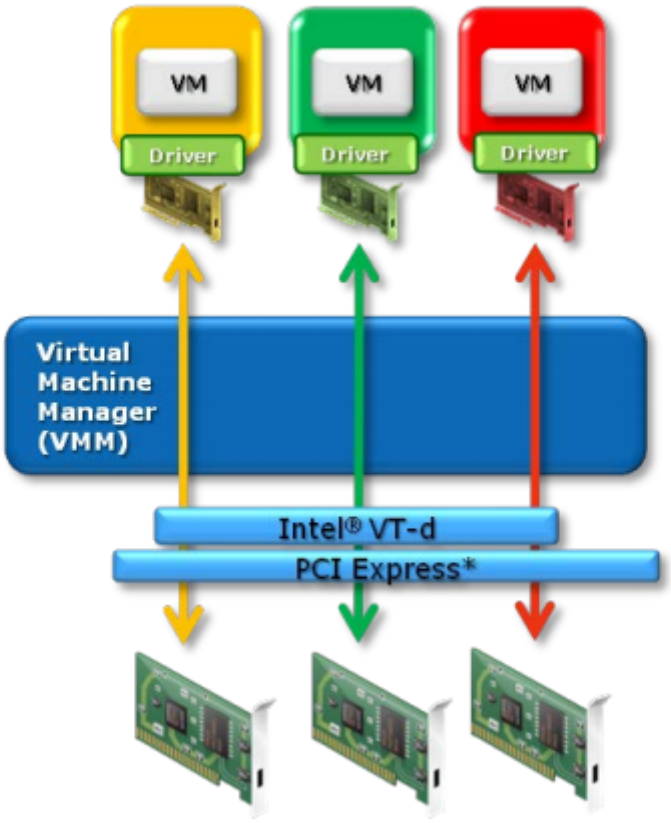


Figure 7. direct assignment

VT-d is Intel's IOMMU implementation. It provides I/O device assignment, DMA remapping, interrupt remapping and other features to improve the performance and security of the virtualization environment.

At least in the context and test environment of this doc, following technical terms refer almost the same thing:

- IOMMU
- VT-d
- PCI device passthgouth
- direct assignment

In simple terms, with VT-d support it's possible to "detach" a device from host, and then attach it into a guest VM. This way the IO performance of the VM can be sharply increased.

In my setup, VT-d was enabled in the IOMMU section as part of system preparation.

the installation script check `/boot/grub/grub.cfg` file to verify whether iommu has been enabled or not.

```
ping@trinity:~$ cat /boot/grub/grub.cfg | grep "intel_iommu=on"
        linux   /vmlinuz-3.19.0-25-generic root=/dev/mapper/trinity--
vg-root ro  intel_iommu=on
                linux   /vmlinuz-3.19.0-25-generic
root=/dev/mapper/trinity--vg-root ro  intel_iommu=on
                linux   /vmlinuz-3.16.0-30-generic
root=/dev/mapper/trinity--vg-root ro  intel_iommu=on
                linux   /vmlinuz-3.13.0-32-generic
root=/dev/mapper/trinity--vg-root ro  intel_iommu=on
```

about VT-d, VT-x, SR-IOV virtualization features

- Intel `VT-d` extensions: or "d"irect-I/O, direct assignment: add virtualization support to **Intel chipsets** that can "assign" specific I/O devices to specific virtual machines (VM)s.
- Intel `VT-x` is virtualization extension specificly on **`CPU`**, while `VT-d` is virtualization on **`chipset`**. VT-x can be thought of as "basic virtualization architecture" in Intel CPU.
- `SR-IOV` is mostly (but "in thoery" not limited to) virtualization on NIC card.

Here is a brief and not-so-acurate summary of each term and its category of virtualization:

term	**virtualization category**
VT-x	CPU
VT-d	chipset
SR-IOV	NIC

16.2. pci_stub

`pci_stub` is a kernel driver module that is used to "hide" the device from host or any other VMs that were not assigned to use this device. This is required to complete the VT-d/PCI-passthough operation.

```
ping@trinity:~$ modinfo pci_stub
filename:       /lib/modules/3.13.0-32-generic/kernel/drivers/pci/pci-stub.ko
author:         Chris Wright <chrisw@sous-sol.org>
license:        GPL
srcversion:     194150416ED68735C4D2803
depends:
intree:         Y
vermagic:       3.13.0-32-generic SMP mod_unload modversions
signer:         Magrathea: Glacier signing key
sig_key:        5E:3C:0F:9C:A6:E3:65:43:53:5F:A2:BB:5B:70:9E:84:F1:6D:A7:C7
sig_hashalgo:   sha512
parm:           ids:Initial PCI IDs to add to the stub driver, format is
                "vendor:device[:subvendor[:subdevice[:class[:class_mask]]]]" and
multiple
                comma separated entries can be specified (string)
```

`pci_stub` was compiled as a kernel module and not to be loaded by default in our case. It can be loaded manually with `modprobe`.

```
ping@trinity:~$ lsmod | grep pci_stub
ping@trinity:~$ modprobe pci_stub
ping@trinity:~$ lsmod | grep pci_stub
pci_stub               12622  0
```

16.3. process to assign PCI device in KVM

Here are the steps to isolate a PCI device, in this example, a SR-IOV VF:

1. before assignment, `p3p1 VF 0` is with `ixgbevf` driver

   ```
   ping@trinity:/images/vmx_20151102.0/build$ sudo lspci -nvks 0000:23:10.0
   23:10.0 0200: 8086:10ed (rev 01)
           Subsystem: 103c:17d2
           Flags: bus master, fast devsel, latency 0
           [virtual] Memory at f3400000 (64-bit, prefetchable) [size=16K]
           [virtual] Memory at f3300000 (64-bit, prefetchable) [size=16K]
           Capabilities: [70] MSI-X: Enable+ Count=3 Masked-
           Capabilities: [a0] Express Endpoint, MSI 00
           Capabilities: [100] Advanced Error Reporting
           Capabilities: [150] Alternative Routing-ID Interpretation (ARI)
           Kernel driver in use: ixgbevf
   ```

   the `-n` option will show PCI vendor and device codes as numbers instead of looking them up in the PCI ID list. In this output it give wendor number **8096:10ed** , which we'll need to use in later commands.

now "unbind" the VF from host, and "bind" it to `pci-stub` driver

```
ping@trinity:/images/vmx_20151102.0/build$ sudo -i
ping@trinity:~# echo "8086 10ed" > /sys/bus/pci/drivers/pci-stub/new_id
root@trinity:~# echo 0000:23:10.0 > /sys/bus/pci/devices/0000:23:10.0/driver/unbind
root@trinity:~# echo 0000:23:10.0 >> /sys/bus/pci/drivers/pci-stub/bind
root@trinity:~# exit
```

these operations requires root privilege to perform.

3. after the "unbind", the `p3p1 VF 0` device is now with pci-stub driver instead of the ixgbevf driver, effectively get "detached" from the host.

```
ping@trinity:/images/vmx_20151102.0/build$ sudo lspci -nvks 0000:23:10.0
23:10.0 0200: 8086:10ed (rev 01)
        Subsystem: 103c:17d2
        Flags: fast devsel
        [virtual] Memory at f3400000 (64-bit, prefetchable) [size=16K]
        [virtual] Memory at f3300000 (64-bit, prefetchable) [size=16K]
        Capabilities: [70] MSI-X: Enable- Count=3 Masked-
        Capabilities: [a0] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [150] Alternative Routing-ID Interpretation (ARI)
        Kernel driver in use: pci-stub      #<------

ping@trinity:/images/vmx_20151102.0/build$ sudo lspci -nvks 0000:06:10.0
06:10.0 0200: 8086:10ed (rev 01)
        Subsystem: 103c:17d2
        Flags: fast devsel
        [virtual] Memory at ee300000 (64-bit, prefetchable) [size=16K]
        [virtual] Memory at ee200000 (64-bit, prefetchable) [size=16K]
        Capabilities: [70] MSI-X: Enable- Count=3 Masked-
        Capabilities: [a0] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [150] Alternative Routing-ID Interpretation (ARI)
        Kernel driver in use: pci-stub      #<------
```

now the `p3p1 VF 0` device is ready to be used in guest VM.

The process to assign a device back from guest VM to host machine is similar - just unbind it from current `pci-stub` driver and then bind it back to `ixgbevf` driver will suffice:

2.

```
ping@trinity:/images/vmx_20151102.0/build$ sudo -i
root@trinity:~# echo "8086 10ed" > /sys/bus/pci/drivers/ixgbevf/new_id
root@trinity:~# echo 0000:23:10.0 > /sys/bus/pci/devices/0000:23:10.0/driver/unbind
root@trinity:~# echo 0000:23:10.0 >> /sys/bus/pci/drivers/ixgbevf/bind
root@trinity:~# exit

ping@trinity:/images/vmx_20151102.0/build$ sudo lspci -nvvs 0000:23:10.0
23:10.0 0200: 8086:10ed (rev 01)
        Subsystem: 103c:17d2
        Flags: bus master, fast devsel, latency 0
        [virtual] Memory at f3400000 (64-bit, prefetchable) [size=16K]
        [virtual] Memory at f3300000 (64-bit, prefetchable) [size=16K]
        Capabilities: [70] MSI-X: Enable+ Count=3 Masked-
        Capabilities: [a0] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [150] Alternative Routing-ID Interpretation (ARI)
        Kernel driver in use: ixgbevf       #<------ back to ixgbevf
```

Chapter 17. SRIOV

17.1. SRIOV basic concept

17.1.1. what is "IO virtualization"?

Input/output (I/O) virtualization is a methodology to simplify management, lower costs and improve performance of servers in enterprise environments. I/O virtualization environments are created by abstracting the upper layer protocols from the physical connections.

SR-IOV is a type of "IO virtualization" technique Developed by the `PCI-SIG` (PCI Special Interest Group), so it is also called `PCI-SIG SR-IOV`.

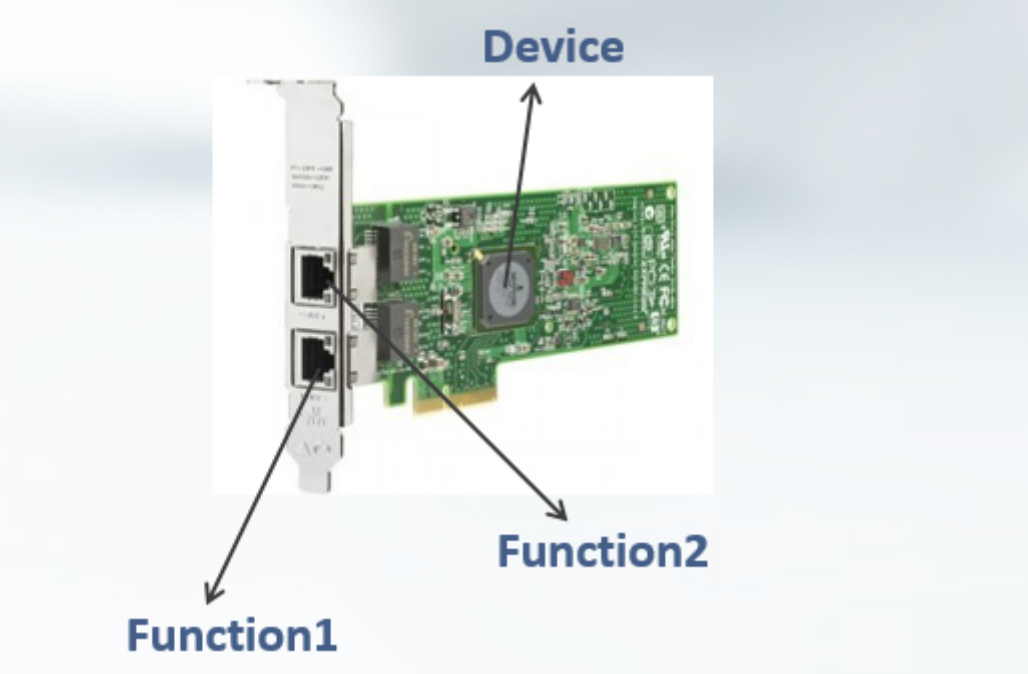
The SR-IOV spec defined a very basic SR-IOV concept: enabling a Single "Root Function" (for example, a single Ethernet port), to appear as multiple, separate, devices. A physical device with SR-IOV capabilities can be configured to appear in the PCI configuration space as multiple "virtual functions".

17.1.2. what is a "function"?

traditionally, a PCIe Device has a Unique PCI Function Address, in the form of `Bus domain` and `Function`, sometimes called a BDF notation, which can be viewed by CLI command `lspci` in a linux system.

```
ping@trinity:~$ lspci | grep -i ethernet
02:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
02:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
```

The last number **0** or **1** here, is called a "function", which usually maps to a physical port:



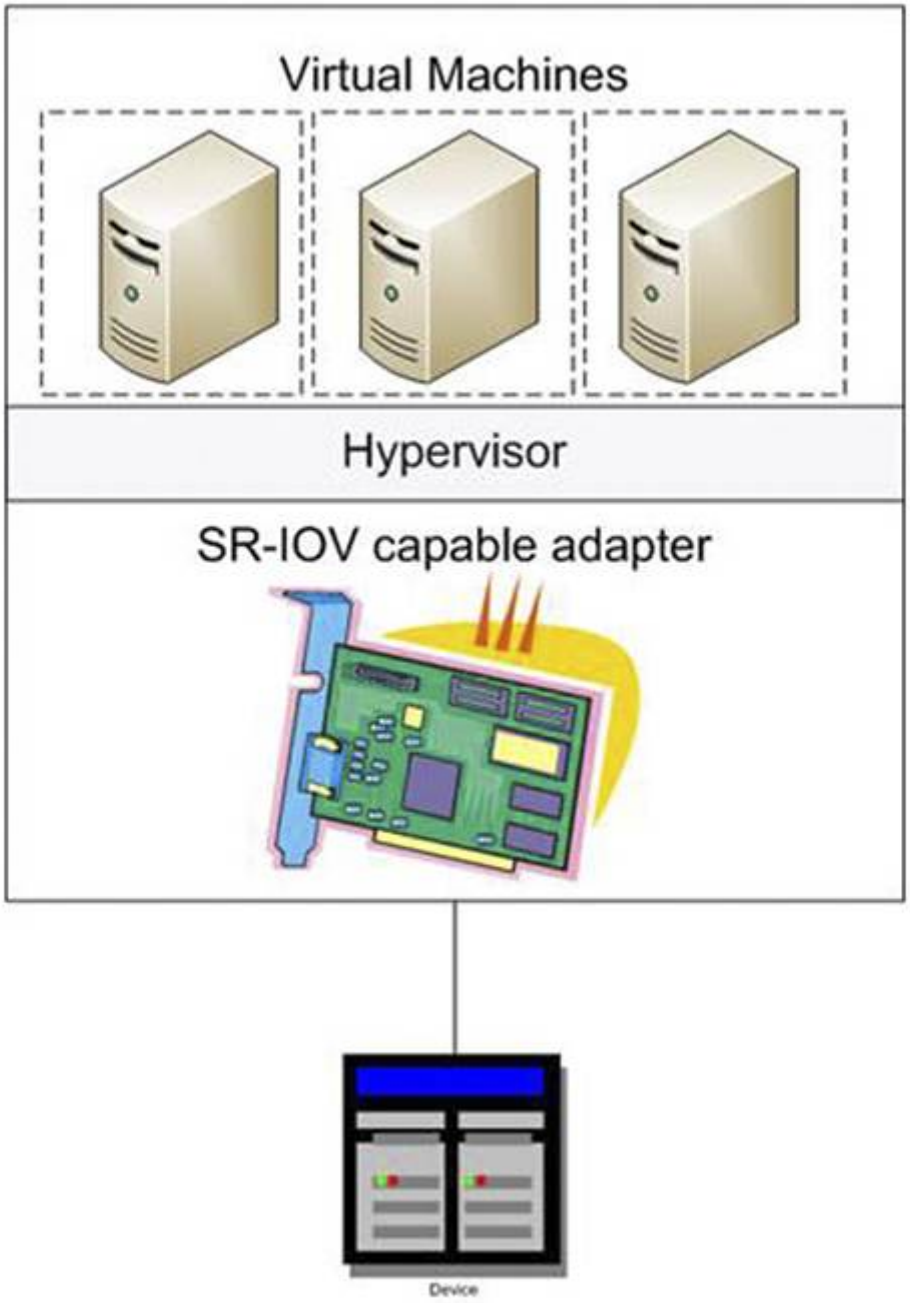
17.1.3. PCI functions defined in SR-IOV

SR-IOV introduces two new function types:

- **Physical Functions (PFs)** are full PCIe devices that include the SR-IOV capabilities. **Physical Functions** are discovered, managed, and configured as normal PCI devices. **Physical Functions** configure and manage the SR-IOV functionality by assigning Virtual Functions.
- **Virtual Functions (VFs)** are simple PCIe functions that only process I/O. These are "lightweight" PCIe functions that contain the resources necessary for data movement but have a carefully minimized set of configuration resources. Each **Virtual Function** is derived from a **Physical Function**. The number of **Virtual Functions** a device may have is limited by the device hardware. A single Ethernet port, the Physical Device, may map to many **Virtual Functions** that can be shared to virtual machines.

The hypervisor can map one or more **Virtual Functions** to a virtual machine. The **Virtual Function** 's configuration space is then mapped to the configuration space presented to the guest.

in another word: with SR-IOV we can "split" one single physical NIC port into multiple "virtual NIC port" , and each virtual NIC port can then be assigned to different VMs.



Each Virtual Function can only be mapped to a single guest at a time, as Virtual Functions require real hardware resources. A virtual machine can have multiple Virtual Functions. A Virtual Function appears as a network card in the same way as a normal network card would appear to an operating system.

The SR-IOV drivers are implemented in the kernel. The core implementation is contained in the PCI subsystem, but there must also be driver support for both the Physical Function (PF) and Virtual Function (VF) devices. An SR-IOV capable device can allocate VFs from a PF. The VFs appear as PCI devices which are backed on the physical PCI device by resources such as queues and register sets. illustrated below:

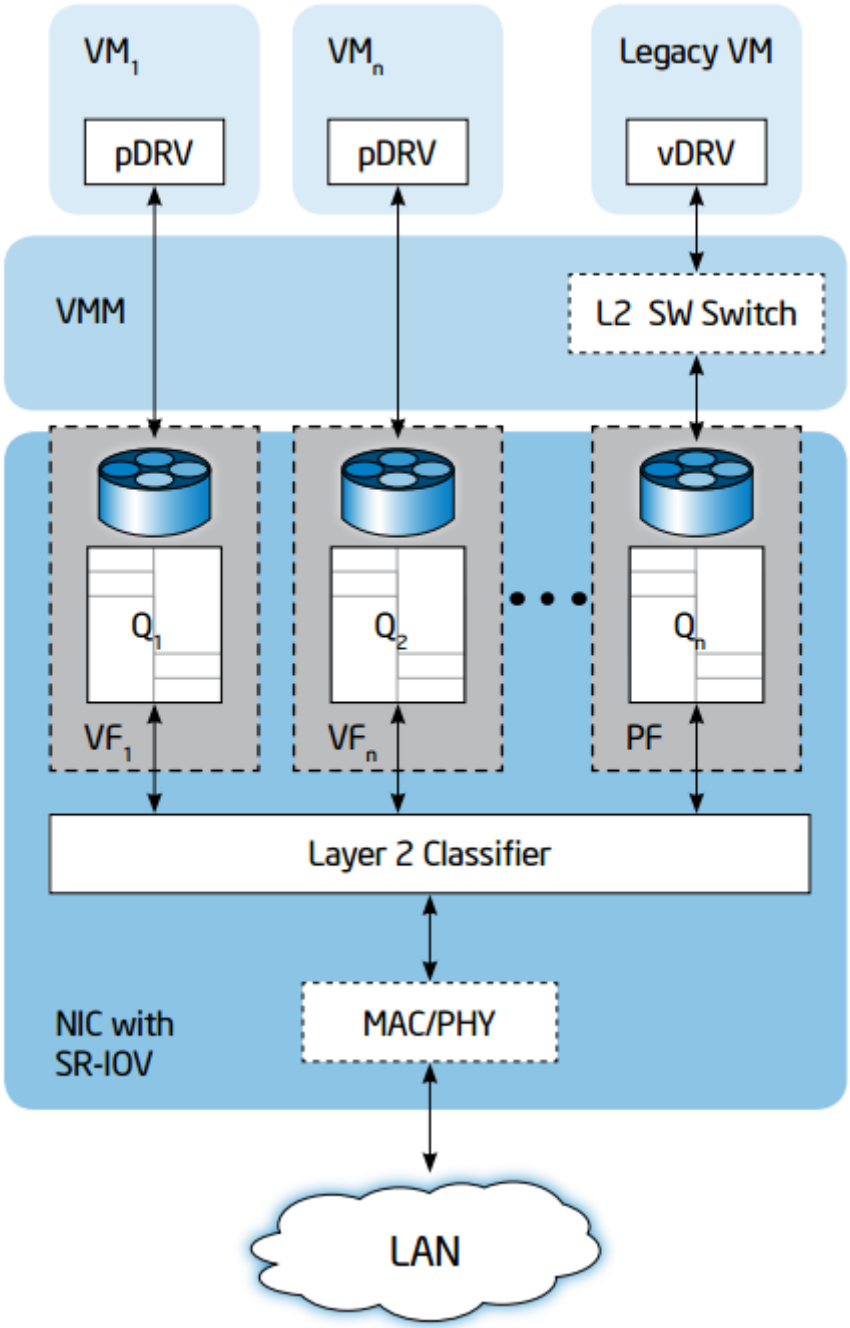


Figure 8. A typical network adapter supporting SR-IOV functionality

- pDRV: Physical Driver(`ixgbe` for Intel 82599 NIC used in our setup)
- vDRV: Virtual Driver(`ixgbevf` for Intel 82599 NIC used in our setup)

17.1.4. what is a "root"?

This is a PCIe term. A `root complex` connects the processor and memory subsystem to the PCIe switch fabric composed of one or more switch devices, similar to a host bridge in a PCI system, which generates transaction requests on behalf of the processor interconnected through a local bus, and may contain more than one PCIe port and multiple switch devices. A root Port is the portion of the motherboard that contains the host bridge, which allows the PCIe ports to talk to the rest of the computer.

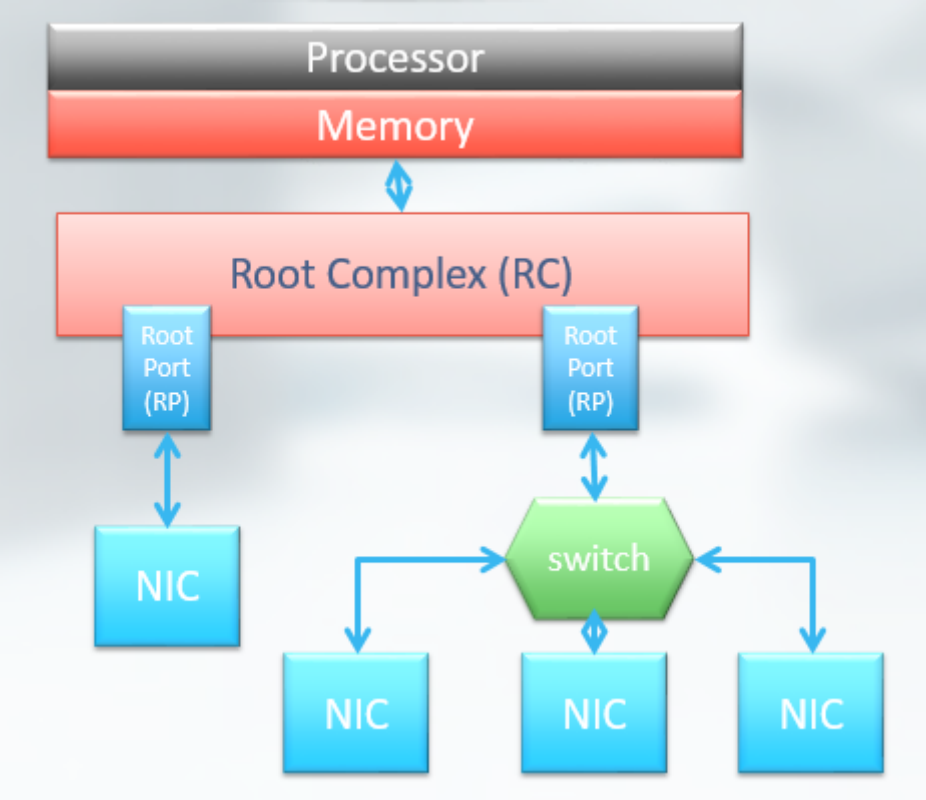
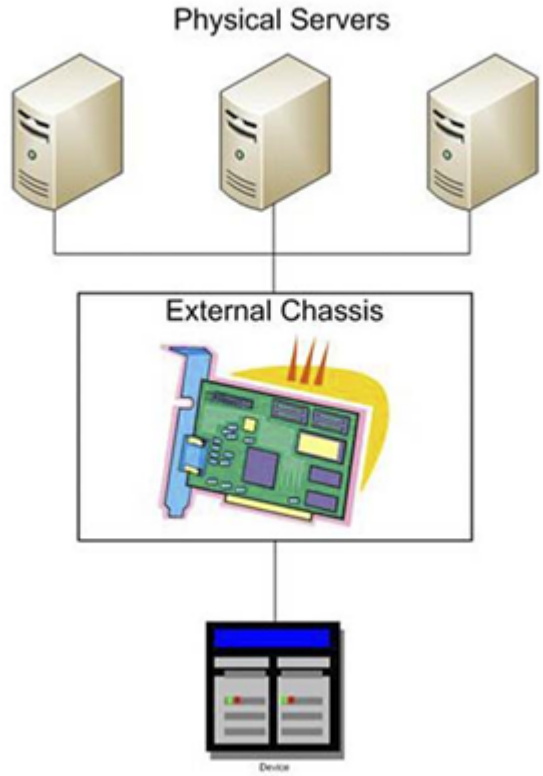


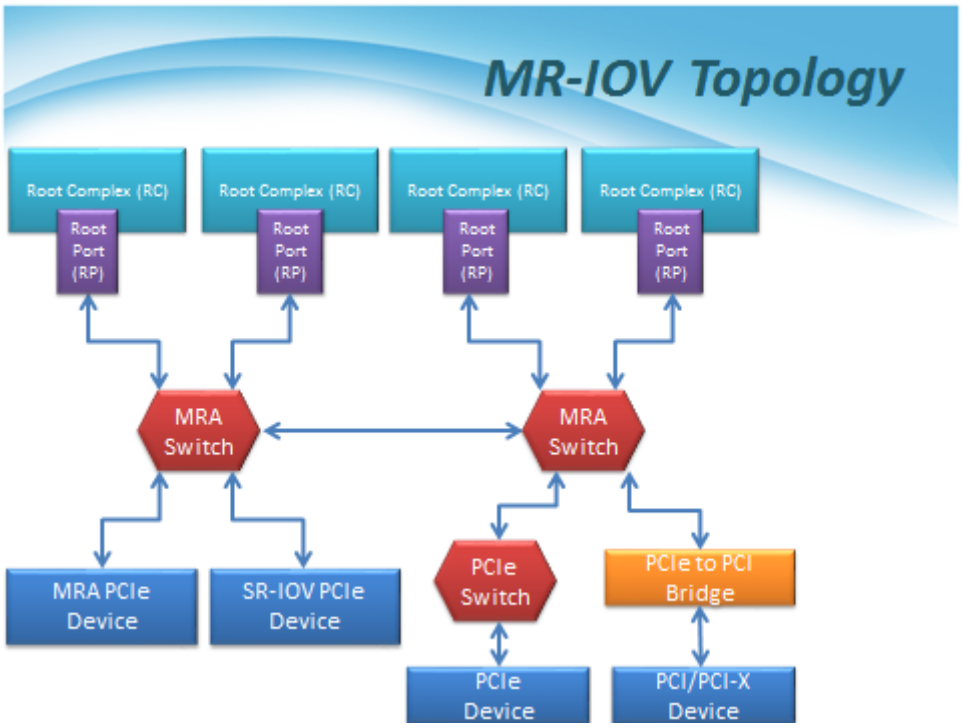
Figure 9. root port

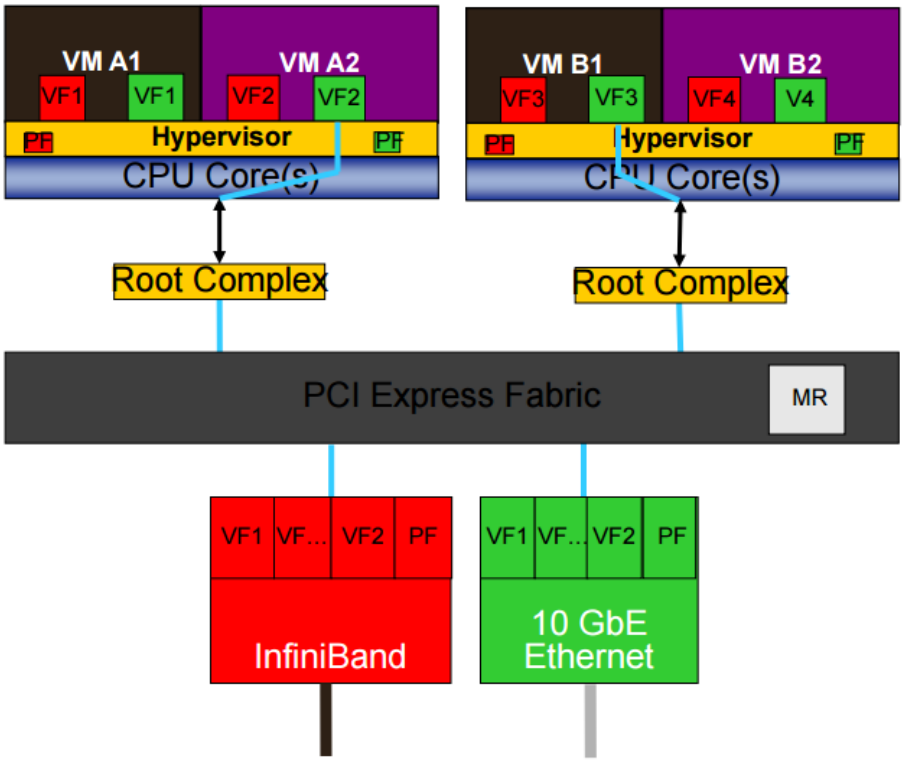
17.1.5. why is it called "single"?

This is to differentiate it with another IO virtualization technology named "MR-IOV" - Multiple Root IOV, which is defined to share IO resource on multiple HW domains. For example, Multiple servers & VMs sharing one I/O adapter, which can be placed into a separate chassis outside of the server. Bandwidth of the I/O adapter is shared among the servers.



a typical MR-IOV topology will look like:



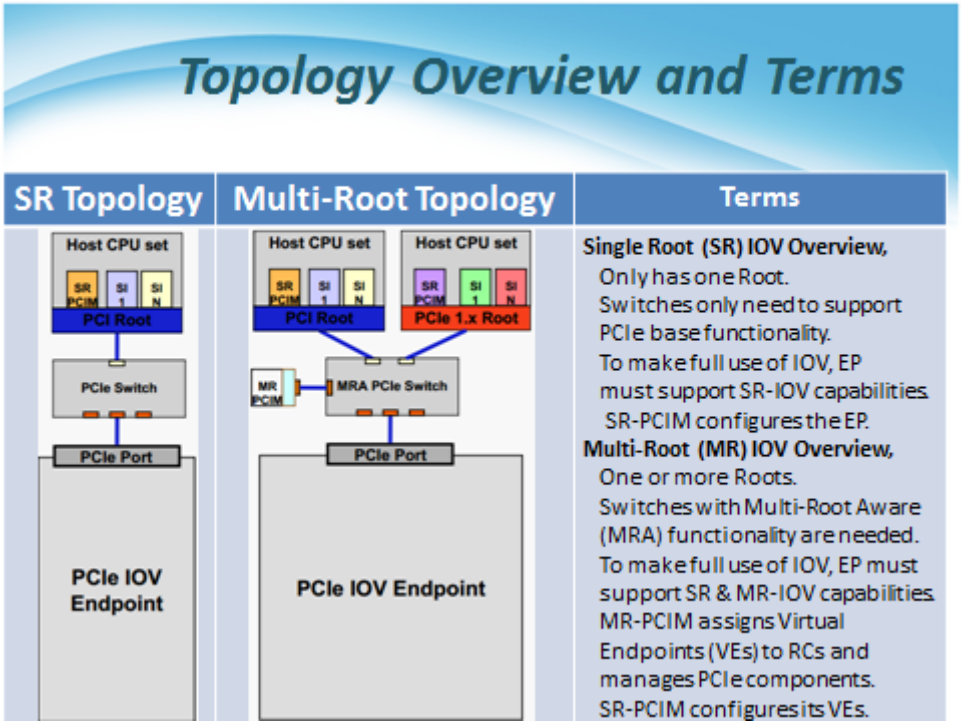


• 10 GbE Controller Receives a Packet for VF2
• Controller determines it goes to VF2 in VM A2
• Controller sends the packet to VM A2
• VM A2 delivers the packet to correct application via Ethernet SW stack

• Application on VM1 sends Ethernet transaction via VF3 to Ethernet Client over 10 GbE
• VM B1 sends out through the Ethernet Controller

• Packet comes in for VF3 B1
• Controller determines that the packet goes to VF3 in VM B1
• Controller sends the packet to VM B1
• VM B1 delivers the packet to correct application via Ethernet SW stack

a brief comparison between SR-IOV and MR-IOV:



17.1.6. corelation with VT-d

Intel's implementation of PCI-SIG SR-IOV functionality requires the VMM software to configure the direct assignment of the virtual function to the virtual machine using Intel® Virtualization Technology for Directed I/O (Intel® VT-d). Memory Translation technologies in Intel® VT-d provide hardware assisted techniques to allow direct DMA transfers. Intel VT-d helps with secure translation, and SRIOV provides separate data spaces for the virtual machines. [6: from SR-IOV spec]

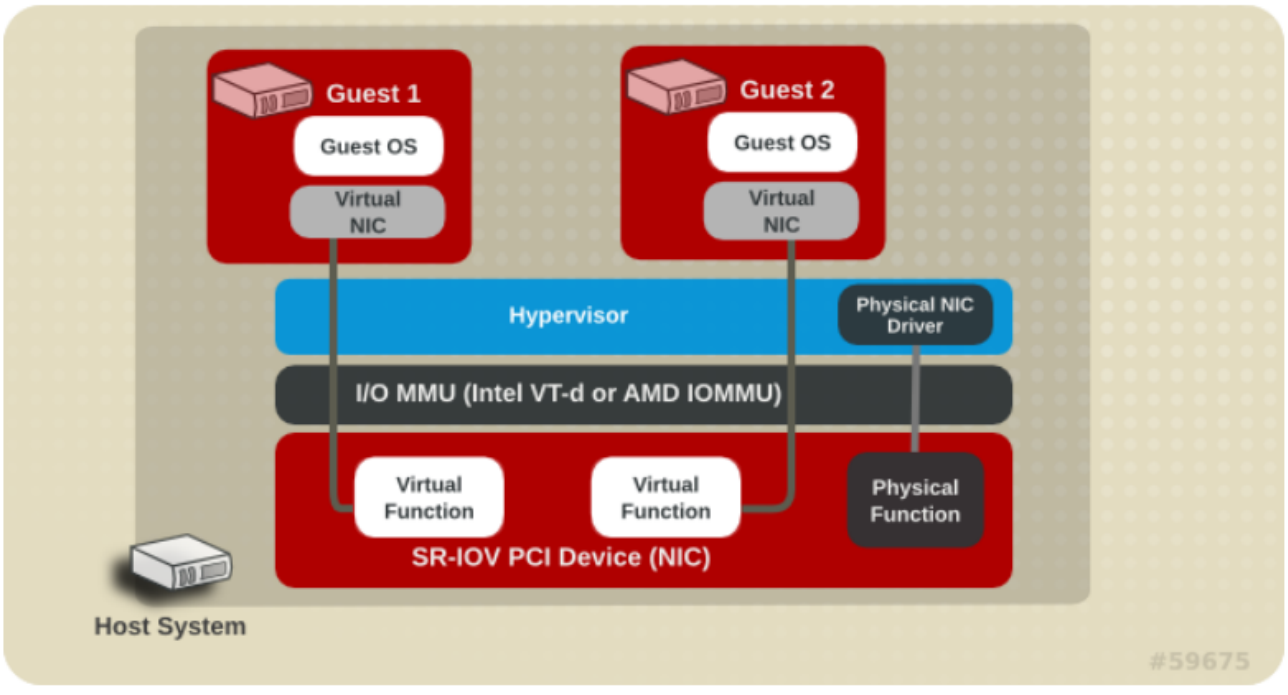


Figure 10. SR-IOV vs. VT-d

In brief, SR-IOV aims to "partition" or "split" the NIC port into VFs, while VT-d aims to "assign" each VF to different VM. If the intention is to always use the entire bandwidth and processing capability

of a NIC, then VT-d can be used without SR-IOV. see section of VT-d for more detailed information about VT-d.

17.1.7. other requirement

currently SR-IOV needs to be supported by different components in a server:

- BIOS support
- HyperVisor support
- NIC and driver support

And, Juniper's modified NIC kernel driver requires linux kernel 3.13. This may be changed later to be compatible with newer kernel releases.

17.2. SR-IOV packet flow

The [SR-IOV-prime] illustrated packet processing flow in SR-IOV:

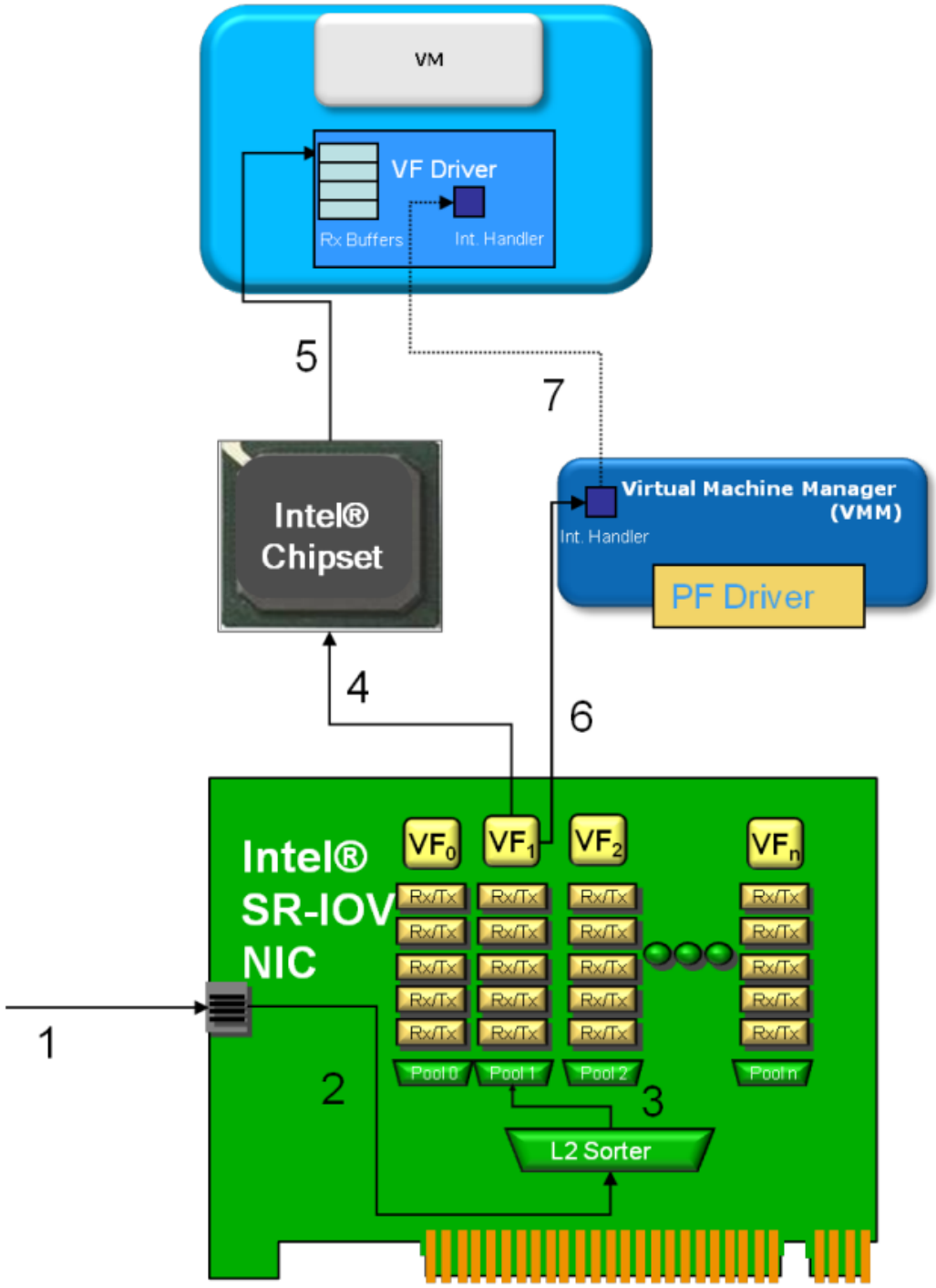


Figure 11. SR-IOV packet flow

1. The Ethernet packet arrives at the Intel® Ethernet NIC.
2. The packet is sent to the Layer 2 sorter/switch/classifier.
   - This Layer 2 sorter is configured by the Master Driver(MD).
   - When either the MD or the VF Driver configure a MAC address or VLAN, this Layer 2 sorter is configured.
3. After being sorted by the Layer 2 Switch, the packet is placed into a receive queue dedicated to the target VF.

For SR-IOV, MAC address configuration in the VF is important. Missing this step will make SR-IOV and VF fail to work, or lead to unexpected behavior.

4. The DMA operation is initiated. The target memory address for the DMA operation is defined within the descriptors in the VF, which have been configured by the VF driver within the VM.

In order for SR-IOV to work, the `ixgbevf` driver kernel module needs to be loaded also in the guest VM.

```
root@localhost:~# lspci | grep 82599
00:06.0 Ethernet controller: Intel Corporation 82599 Ethernet
Controller Virtual Function (rev 01)
00:07.0 Ethernet controller: Intel Corporation 82599 Ethernet
Controller Virtual Function (rev 01)
00:08.0 Ethernet controller: Intel Corporation 82599 Ethernet
Controller Virtual Function (rev 01)
00:09.0 Ethernet controller: Intel Corporation 82599 Ethernet
Controller Virtual Function (rev 01)

root@localhost:~# lspci -vks 00:06.0
00:06.0 Ethernet controller: Intel Corporation 82599 Ethernet
Controller Virtual Function (rev 01)
        Subsystem: Hewlett-Packard Company Device 17d3
        Flags: bus master, fast devsel, latency 0
        Memory at fe000000 (64-bit, prefetchable) [size=16K]
        Memory at fe004000 (64-bit, prefetchable) [size=16K]
        Capabilities: [a0] Express Endpoint, MSI 00
        Capabilities: [70] MSI-X: Enable+ Count=3 Masked-
        Kernel driver in use: igb_uio
        Kernel modules: ixgbevf
```

5. The DMA Operation has reached the chipset. Intel® VT-d, which has been configured by the VMM then remaps the target DMA address from a virtual host address to a physical host address. The DMA operation is completed; the Ethernet packet is now in the memory space of the VM.

As mentioned earlier, Intel's SR-IOV implementation requires VT-d to be enabled.

6. The Intel® Ethernet NIC fires an interrupt, indicating a packet has arrived. This interrupt is handled by the VMM.
7. The VMM fires a virtual interrupt to the VM, so that it is informed that the packet has arrived.

this is why sometime it is called "interupt-driven" work mode.

17.3. SR-IOV configuration in VMX

in `vmx.conf` file, the `device-type : sriov` option will instruct the installation script to configure `Virtual Function` (VF) on NIC card.

the actual CLI commands executed by the script to configure `SR-IOV` is very simple: just reload the ixgbe module with `max_vfs` parameter:

```
sudo rmmod ixgbe;
sudo modprobe ixgbe max_vfs=1,1,1,1,1,1,1,1;
```

just change the `max_vfs` parameter (here is **1**) to the needed number of VF in your setup.

In practice, There is a small problem here:

- The first command `rmmod ixgbe` will remove the kernel driver module
- as a result all interfaces driven by the ixgbe module will be removed from the platform as well.
- In the case that the mgmt port is also ixgbe-driven, This will disconnect the current telnet/ssh session and we'll need to login via console (e.g through `ilo`).

To workaround this and also to reduce the time of interuption to our telnet/ssh session, I prefer to use below concatenated commands to configure SR-IOV FVs immediately right after ixgbe module removed, and add our mgmt interface back to the external bridge, etc, all in one go.

```
sudo rmmod ixgbevf; sudo rmmod ixgbe; \
sudo modprobe ixgbe max_vfs=1,1,1,1,1,1,1,1; \
sudo modprobe tun ; sleep 5; \
sudo brctl addif br-ext em1
```

Below `interface` configuration in `vmx.conf` will specify the mapping between VF in the host and NIC in the guest VM, and properties of the VF.

```
  - interface            : ge-0/0/0
    port-speed-mbps      : 10000
    nic                  : p3p1
    mtu                  : 2000
    virtual-function     : 0
    mac-address          : "02:04.17:01:02:01"
    description          : "ge-0/0/0 connects to eth6"

  - interface            : ge-0/0/1
    port-speed-mbps      : 10000
    nic                  : p2p1
    mtu                  : 2000
    virtual-function     : 0
    mac-address          : "02:04.17:01:02:02"
    description          : "ge-0/0/1 connects to eth7"
```

17.4. example of 1 VF

to list all PF and VFs, use `lspci`

```
ping@trinity:~$ lspci | grep -i "ethernet"
02:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
02:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
02:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
02:10.1 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
06:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
06:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
06:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
06:10.1 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
21:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
21:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
21:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
21:10.1 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
23:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
23:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
23:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
23:10.1 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
```

kernel logs will display how many VFs have been enabled for each and every port.

kernel logs:

```
ping@trinity:~$ dmesg | grep SR-IOV
[203216.832237] ixgbe 0000:02:00.0 (unregistered net_device): SR-IOV enabled with 1
VFs
[203217.067941] ixgbe 0000:02:00.1 (unregistered net_device): SR-IOV enabled with 1
VFs
[203217.303815] ixgbe 0000:06:00.0 (unregistered net_device): SR-IOV enabled with 1
VFs
[203217.539589] ixgbe 0000:06:00.1 (unregistered net_device): SR-IOV enabled with 1
VFs
[203217.783265] ixgbe 0000:21:00.0 (unregistered net_device): SR-IOV enabled with 1
VFs
[203218.022982] ixgbe 0000:21:00.1 (unregistered net_device): SR-IOV enabled with 1
VFs
[203218.254658] ixgbe 0000:23:00.0 (unregistered net_device): SR-IOV enabled with 1
VFs
[203218.490378] ixgbe 0000:23:00.1 (unregistered net_device): SR-IOV enabled with 1
VFs
```

One way to find out PCI bus address of an interface name, is to use `ethtool` with `-i` option:

```
ping@trinity:~$ ethtool -i p2p1
driver: ixgbe
version: 3.19.1
firmware-version: 0x8000076f, 1.475.0
bus-info: 0000:06:00.0  #<------
supports-statistics: yes
supports-test: yes
supports-eeprom-access: yes
supports-register-dump: yes
supports-priv-flags: no

ping@trinity:~$ ethtool -i p3p1
driver: ixgbe
version: 3.19.1
firmware-version: 0x8000076f, 1.475.0
bus-info: 0000:23:00.0  #<------
supports-statistics: yes
supports-test: yes
supports-eeprom-access: yes
supports-register-dump: yes
supports-priv-flags: no
ping@trinity:~$
```

The other option is to use `lshw` command, which can be used to list vendor/manufacturer info of all interfaces in the platform, an example is shown below:

```
ping@ubuntu1:~$ sudo lshw -class network | grep -iE "bus info|logical
name"
       bus info: pci@0000:07:00.0
       logical name: p2p1
       bus info: pci@0000:07:00.1
       logical name: p2p2
       bus info: pci@0000:07:10.0
       bus info: pci@0000:07:10.1
       bus info: pci@0000:07:10.2
       bus info: pci@0000:07:10.3
       bus info: pci@0000:07:10.4
       bus info: pci@0000:07:10.5
       bus info: pci@0000:07:10.6
       bus info: pci@0000:07:10.7
       bus info: pci@0000:03:00.0
       logical name: em1
       bus info: pci@0000:03:00.1
       logical name: em2
       bus info: pci@0000:03:00.2
       logical name: em3
       bus info: pci@0000:03:00.3
       logical name: em4
       bus info: pci@0000:0a:00.0
       logical name: p3p1
       bus info: pci@0000:0a:00.1
       logical name: p3p2
       bus info: pci@0000:24:00.0
       logical name: p5p1
       bus info: pci@0000:24:00.1
       logical name: p5p2
       bus info: pci@0000:24:10.0
       bus info: pci@0000:24:10.1
       bus info: pci@0000:24:10.2
       bus info: pci@0000:24:10.3
       bus info: pci@0000:24:10.4
       bus info: pci@0000:24:10.5
       bus info: pci@0000:24:10.6
       bus info: pci@0000:24:10.7
       logical name: br-int-vmx1-nic
       logical name: vfp_int-vmx1
       logical name: vfp_ext-vmx1
       logical name: br-ext-nic
       logical name: vcp_int-vmx1
       logical name: vcp_ext-vmx1
```

the advantage of `lshw` over `ethtool` is, it list all network interfaces hardware information instead of just one specific interfac, and, it also shows the mapping relationship between PCI address and interface name, which comes very handy.

with the bus address, the PF-VF mapping relationship can be printed:

```
ping@trinity:~$ ls -l /sys/bus/pci/devices/0000\:06\:00.0/virtfn*
lrwxrwxrwx 1 root root 0 Nov 23 21:03 /sys/bus/pci/devices/0000:06:00.0/virtfn0 ->
../0000:06:10.0

ping@trinity:~$ ls -l /sys/bus/pci/devices/0000\:23\:00.0/virtfn*
lrwxrwxrwx 1 root root 0 Nov 23 21:03 /sys/bus/pci/devices/0000:23:00.0/virtfn0 ->
../0000:23:10.0
```

to get all PF-VF map in a platform, "find" it like this:

```
ping@trinity:~$ sudo find /sys -name virtfn* | xargs ls -l
... /sys/.../0000:02:00.0/virtfn0 -> ../0000:02:10.0
... /sys/.../0000:02:00.1/virtfn0 -> ../0000:02:10.1
... /sys/.../0000:06:00.0/virtfn0 -> ../0000:06:10.0
... /sys/.../0000:06:00.1/virtfn0 -> ../0000:06:10.1
... /sys/.../0000:21:00.0/virtfn0 -> ../0000:21:10.0
... /sys/.../0000:21:00.1/virtfn0 -> ../0000:21:10.1
... /sys/.../0000:23:00.0/virtfn0 -> ../0000:23:10.0
... /sys/.../0000:23:00.1/virtfn0 -> ../0000:23:10.1
```

another method to find out VF-PF mapping corelation is to use libvirt `virsh` command:

list running VMs and locate VCP name/ID:

```
ping@ubuntu1:~$ sudo virsh list
 Id    Name                           State
----------------------------------------------------
 34    vhepe-vcp                      running
 37    vhepe-vfp                      running
```

list VFs(normally with slot number '0x10' or '0x11') in use:

```
ping@ubuntu1:~$ sudo virsh dumpxml 37 | grep -iE "0x10|0x11"
        <address type='pci' domain='0x0000' bus='0x24' slot='0x10'
function='0x0'/>
        <address type='pci' domain='0x0000' bus='0x24' slot='0x10'
function='0x1'/>
        <address type='pci' domain='0x0000' bus='0x07' slot='0x10'
function='0x0'/>
        <address type='pci' domain='0x0000' bus='0x07' slot='0x10'
function='0x1'/>
```

locate the nodedev name and print the mapping relationship

```
ping@ubuntu1:~$ sudo virsh nodedev-list | grep 0000_24_10
pci_0000_24_10_0
pci_0000_24_10_1
pci_0000_24_10_2
pci_0000_24_10_3
pci_0000_24_10_4
pci_0000_24_10_5
pci_0000_24_10_6
pci_0000_24_10_7

ping@ubuntu1:~$ sudo virsh nodedev-dumpxml pci_0000_24_10_0 | grep
function
    <function>0</function>
    <capability type='phys_function'>
      <address domain='0x0000' bus='0x24' slot='0x00' function='0x0'/>
①
      <address domain='0x0000' bus='0x24' slot='0x10' function='0x0'/>
②
```

① PF address

② VF address

if we put all info collected from above commands, we can produce a table that lists the mapping relationship between PF, VF PCI bus address, logical interface name, and the VMX (guest VM) virtual interface name, like this:

Table 5. NIC mapping table (1 VF)

PCI address	adapter	interface	VF address	VF#	VMX interface
02:00.0	560FLB	em9	02:10.0	0	
02:00.1	560FLB	em10	02:10.1	0	
06:00.0	560M	p2p1	06:10.0	0	ge-0/0/1
06:00.1	560M	p2p2	06:10.1	0	
21:00.0	560FLB	em1	21:10.0	0	
21:00.1	560FLB	em2	21:10.1	0	ge-0/0/0
23:00.0	560M	p3p1	23:10.0	0	
23:00.1	560M	p3p2	23:10.1	0	

It will be very handy to have this table in hand, before you proceed to plan your VMX installation. It will become even helpful if you plan to setup a multi-instances VMX lab involving multiple VFs being configured and used. In below sections I'll demonstrate examples of configuring 4 VFs and 8 VFs.

17.5. example of 4 VFs

To enable maximum of 4 VFs, give `max_vfs` number of **4** instead of **1** and repeat the same exact commands:

```
sudo rmmod ixgbevf; sudo rmmod ixgbe; \
sudo modprobe ixgbe max_vfs=4,4,4,4,4,4,4,4; \
sudo modprobe tun ; sleep 5; \
sudo brctl addif br-ext em1
```

list all generated VFs:

```
ping@trinity:~$ lspci | grep -i ethernet
02:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
02:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
02:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
...<snippet>...
02:10.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
06:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
06:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
06:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
06:10.1 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
06:10.3 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
06:10.5 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
06:10.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
21:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
21:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
21:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
...<snippet>...
21:10.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
23:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
23:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
23:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
23:10.1 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
23:10.3 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
23:10.5 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
23:10.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
```

the kernel logs (dmesg) now show 4 VFs enabled

```
[26029.301160] ixgbe 0000:02:00.0 (unregistered net_device): SR-IOV enabled with 4 VFs
[26029.540597] ixgbe 0000:02:00.1 (unregistered net_device): SR-IOV enabled with 4 VFs
[26029.671906] ixgbe 0000:06:00.0 (unregistered net_device): SR-IOV enabled with 1 VFs
[26029.908841] ixgbe 0000:06:00.1 (unregistered net_device): SR-IOV enabled with 4 VFs
[26030.144053] ixgbe 0000:21:00.0 (unregistered net_device): SR-IOV enabled with 4 VFs
[26030.383774] ixgbe 0000:21:00.1 (unregistered net_device): SR-IOV enabled with 4 VFs
[26030.515073] ixgbe 0000:23:00.0 (unregistered net_device): SR-IOV enabled with 1 VFs
[26030.752200] ixgbe 0000:23:00.1 (unregistered net_device): SR-IOV enabled with 4 VFs
```

Not all NIC ports now have 4 VFs , which will be explained later.

number of VFs currently enabled, vs number of maximum VFs supported, per NIC port:

```
ping@trinity:~$ sudo find /sys/ -name "*vfs*"
/sys/devices/pci0000:00/0000:00:02.2/0000:02:00.0/sriov_numvfs
/sys/devices/pci0000:00/0000:00:02.2/0000:02:00.0/sriov_totalvfs
/sys/devices/pci0000:00/0000:00:02.2/0000:02:00.1/sriov_numvfs
/sys/devices/pci0000:00/0000:00:02.2/0000:02:00.1/sriov_totalvfs
/sys/devices/pci0000:00/0000:00:03.0/0000:06:00.0/sriov_numvfs
/sys/devices/pci0000:00/0000:00:03.0/0000:06:00.0/sriov_totalvfs
/sys/devices/pci0000:00/0000:00:03.0/0000:06:00.1/sriov_numvfs
/sys/devices/pci0000:00/0000:00:03.0/0000:06:00.1/sriov_totalvfs
/sys/devices/pci0000:20/0000:20:02.2/0000:21:00.0/sriov_numvfs
/sys/devices/pci0000:20/0000:20:02.2/0000:21:00.0/sriov_totalvfs
/sys/devices/pci0000:20/0000:20:02.2/0000:21:00.1/sriov_numvfs
/sys/devices/pci0000:20/0000:20:02.2/0000:21:00.1/sriov_totalvfs
/sys/devices/pci0000:20/0000:20:03.0/0000:23:00.0/sriov_numvfs
/sys/devices/pci0000:20/0000:20:03.0/0000:23:00.0/sriov_totalvfs
/sys/devices/pci0000:20/0000:20:03.0/0000:23:00.1/sriov_numvfs
/sys/devices/pci0000:20/0000:20:03.0/0000:23:00.1/sriov_totalvfs
/sys/kernel/debug/tracing/events/vfs

ping@trinity:~$ sudo cat
/sys/devices/pci0000:00/0000:00:02.2/0000:02:00.0/sriov_numvfs
4
ping@trinity:~$ sudo cat
/sys/devices/pci0000:00/0000:00:02.2/0000:02:00.0/sriov_totalvfs
63
ping@trinity:~$ sudo cat
/sys/devices/pci0000:00/0000:00:03.0/0000:06:00.0/sriov_numvfs
1
ping@trinity:~$ sudo cat
/sys/devices/pci0000:00/0000:00:03.0/0000:06:00.0/sriov_totalvfs
63
ping@trinity:~$ sudo cat
/sys/devices/pci0000:00/0000:00:03.0/0000:06:00.1/sriov_numvfs
4
ping@trinity:~$ sudo cat
```

```
/sys/devices/pci0000:00/0000:00:03.0/0000:06:00.1/sriov_totalvfs
63
ping@trinity:~$ sudo cat
/sys/devices/pci0000:20/0000:20:02.2/0000:21:00.0/sriov_numvfs
4
ping@trinity:~$ sudo cat
/sys/devices/pci0000:20/0000:20:02.2/0000:21:00.0/sriov_totalvfs
63
ping@trinity:~$ sudo cat
/sys/devices/pci0000:20/0000:20:02.2/0000:21:00.1/sriov_numvfs
4
ping@trinity:~$ sudo cat
/sys/devices/pci0000:20/0000:20:02.2/0000:21:00.1/sriov_totalvfs
63
ping@trinity:~$ sudo cat
/sys/devices/pci0000:20/0000:20:03.0/0000:23:00.0/sriov_numvfs
1
ping@trinity:~$ sudo cat
/sys/devices/pci0000:20/0000:20:03.0/0000:23:00.0/sriov_totalvfs
63
ping@trinity:~$ sudo cat
/sys/devices/pci0000:20/0000:20:03.0/0000:23:00.1/sriov_numvfs
4
ping@trinity:~$ sudo cat
/sys/devices/pci0000:20/0000:20:03.0/0000:23:00.1/sriov_totalvfs
63
```

VF-PF mapping:

```
ping@trinity:~$ ls -l /sys/bus/pci/devices/0000\:02\:00.0/virtfn*
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:02:00.0/virtfn0 ->
../0000:02:10.0
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:02:00.0/virtfn1 ->
../0000:02:10.2
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:02:00.0/virtfn2 ->
../0000:02:10.4
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:02:00.0/virtfn3 ->
../0000:02:10.6
ping@trinity:~$

ping@trinity:~$ ls -l /sys/bus/pci/devices/0000\:02\:00.1/virtfn*
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:02:00.1/virtfn0 ->
../0000:02:10.1
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:02:00.1/virtfn1 ->
../0000:02:10.3
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:02:00.1/virtfn2 ->
../0000:02:10.5
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:02:00.1/virtfn3 ->
../0000:02:10.7
ping@trinity:~$
```

```
ping@trinity:~$ ls -l /sys/bus/pci/devices/0000\:06\:00.0/rtfn*
lrwxrwxrwx 1 root root 0 Nov 21 11:57 /sys/bus/pci/devices/0000:06:00.0/virtfn0 ->
../0000:06:10.0

ping@trinity:~$ ls -l /sys/bus/pci/devices/0000\:06:00.1/virtfn*
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:06:00.1/virtfn0 ->
../0000:06:10.1
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:06:00.1/virtfn1 ->
../0000:06:10.3
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:06:00.1/virtfn2 ->
../0000:06:10.5
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:06:00.1/virtfn3 ->
../0000:06:10.7

ping@trinity:~$ ls -l /sys/bus/pci/devices/0000\:21:00.0/virtfn*
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:21:00.0/virtfn0 ->
../0000:21:10.0
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:21:00.0/virtfn1 ->
../0000:21:10.2
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:21:00.0/virtfn2 ->
../0000:21:10.4
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:21:00.0/virtfn3 ->
../0000:21:10.6

ping@trinity:~$ ls -l /sys/bus/pci/devices/0000\:21:00.1/virtfn*
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:21:00.1/virtfn0 ->
../0000:21:10.1
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:21:00.1/virtfn1 ->
../0000:21:10.3
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:21:00.1/virtfn2 ->
../0000:21:10.5
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:21:00.1/virtfn3 ->
../0000:21:10.7

ping@trinity:~$ ls -l /sys/bus/pci/devices/0000\:23:00.0/virtfn*
lrwxrwxrwx 1 root root 0 Nov 21 11:57 /sys/bus/pci/devices/0000:23:00.0/virtfn0 ->
../0000:23:10.0

ping@trinity:~$ ls -l /sys/bus/pci/devices/0000\:23:00.1/virtfn*
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:23:00.1/virtfn0 ->
../0000:23:10.1
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:23:00.1/virtfn1 ->
../0000:23:10.3
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:23:00.1/virtfn2 ->
../0000:23:10.5
lrwxrwxrwx 1 root root 0 Nov 21 18:22 /sys/bus/pci/devices/0000:23:00.1/virtfn3 ->
../0000:23:10.7
```

the PCI bus address may look different on different server, so correct PCI address

has to be used to execute each command above. A more convenient and general way to display all PF-VF mapping is to `find` them out under `/sys` folder. below is a sample captured from a different server.

```
ping@ubuntu1:~$ sudo find /sys -name virtfn* | xargs ls -l
... /sys/devices/pci0000:00/0000:00:01.0/0000:07:00.0/virtfn0 ->
../0000:07:10.0
... /sys/devices/pci0000:00/0000:00:01.0/0000:07:00.0/virtfn1 ->
../0000:07:10.2
... /sys/devices/pci0000:00/0000:00:01.0/0000:07:00.0/virtfn2 ->
../0000:07:10.4
... /sys/devices/pci0000:00/0000:00:01.0/0000:07:00.0/virtfn3 ->
../0000:07:10.6
... /sys/devices/pci0000:00/0000:00:01.0/0000:07:00.1/virtfn0 ->
../0000:07:10.1
... /sys/devices/pci0000:00/0000:00:01.0/0000:07:00.1/virtfn1 ->
../0000:07:10.3
... /sys/devices/pci0000:00/0000:00:01.0/0000:07:00.1/virtfn2 ->
../0000:07:10.5
... /sys/devices/pci0000:00/0000:00:01.0/0000:07:00.1/virtfn3 ->
../0000:07:10.7
... /sys/devices/pci0000:20/0000:20:02.2/0000:24:00.0/virtfn0 ->
../0000:24:10.0
... /sys/devices/pci0000:20/0000:20:02.2/0000:24:00.0/virtfn1 ->
../0000:24:10.2
... /sys/devices/pci0000:20/0000:20:02.2/0000:24:00.0/virtfn2 ->
../0000:24:10.4
... /sys/devices/pci0000:20/0000:20:02.2/0000:24:00.0/virtfn3 ->
../0000:24:10.6
... /sys/devices/pci0000:20/0000:20:02.2/0000:24:00.1/virtfn0 ->
../0000:24:10.1
... /sys/devices/pci0000:20/0000:20:02.2/0000:24:00.1/virtfn1 ->
../0000:24:10.3
... /sys/devices/pci0000:20/0000:20:02.2/0000:24:00.1/virtfn2 ->
../0000:24:10.5
... /sys/devices/pci0000:20/0000:20:02.2/0000:24:00.1/virtfn3 ->
../0000:24:10.7
```

Similiarly, to print the number of currently enabled VFs on each interfaces:

```
ping@ubuntu1:/export/home/vhepe/images$ sudo find /sys/ -name
"*numvfs" | xargs cat
4
4
0
0
4
4
```

to print the capability of support maximum VFs on each port:

```
ping@ubuntu1:/export/home/vhepe/images$ sudo find /sys/ -name
"*totalvfs" | xargs cat
64
64
64
64
64
64
```

Table 6. NIC mapping table (4 VFs)

PCI address	adapter	interface	VF address	VF#	VMX interface
02:00.0	560FLB	em9	02:10.0	0	
			02:10.2	1	
			02:10.4	2	
			02:10.6	3	
02:00.1	560FLB	em10	02:10.1	0	
			02:10.3	1	
			02:10.5	2	
			02:10.7	3	
06:00.0	560M	p2p1	06:10.0	0	ge-0/0/1
06:00.1	560M	p2p2	06:10.1	0	
			06:10.3	1	
			06:10.5	2	
			06:10.7	3	
21:00.0	560FLB	em1	21:10.0	0	
			21:10.2	1	
			21:10.4	2	
			21:10.6	3	
21:00.1	560FLB	em2	21:10.1	0	
			21:10.3	1	
			21:10.5	2	
			21:10.7	3	
23:00.0	560M	p3p1	23:10.0	0	ge-0/0/0

PCI address	adapter	interface	VF address	VF#	VMX interface
23:00.1	560M	p3p2	23:10.1	0	
			23:10.3	1	
			23:10.5	2	
			23:10.7	3	

Now, the reason that `p2p1` and `p3p1` still hold just 1 VF instead of 4 in this example, is because the guest VMX VMs were not torn down at the time when SR-IOV VFs were reconfigured, so the previous VF resources were still hold unchanged on the ports in use.

After removing the VMs and reconfiguring ixgbe the expected number of VFs can be seen:

```
sudo virsh destroy vcp-vmx1
sudo virsh destroy vfp-vmx1
sudo virsh undefine vcp-vmx1
sudo virsh undefine vfp-vmx1
sudo rmmod ixgbevf; sudo rmmod ixgbe; \
sudo modprobe ixgbe max_vfs=4,4,4,4,4,4,4,4; \
sudo modprobe tun ; sleep 5; sudo brctl addif em1

[206499.970842] ixgbe 0000:02:00.0 (unregistered net_device): SR-IOV enabled with 4
VFs
[206500.206573] ixgbe 0000:02:00.1 (unregistered net_device): SR-IOV enabled with 4
VFs
[206500.446766] ixgbe 0000:06:00.0 (unregistered net_device): SR-IOV enabled with 4
VFs
[206500.682536] ixgbe 0000:06:00.1 (unregistered net_device): SR-IOV enabled with 4
VFs
[206500.917846] ixgbe 0000:21:00.0 (unregistered net_device): SR-IOV enabled with 4
VFs
[206501.153595] ixgbe 0000:21:00.1 (unregistered net_device): SR-IOV enabled with 4
VFs
[206501.389801] ixgbe 0000:23:00.0 (unregistered net_device): SR-IOV enabled with 4
VFs
[206501.625554] ixgbe 0000:23:00.1 (unregistered net_device): SR-IOV enabled with 4
VFs
```

17.5.1. the "rule of thumb" for 4 VF

The previous table for "4 VFs" scenario will now be updated as below:

Table 7. NIC mapping table

PCI address	adapter	interface	VF address	VF#	VMX interface
02:00.0	560FLB	em9	02:10.0	0	
			02:10.2	1	
			02:10.4	2	
			02:10.6	3	
02:00.1	560FLB	em10	02:10.1	0	
			02:10.3	1	
			02:10.5	2	
			02:10.7	3	
06:00.0	560M	p2p1	06:10.0	0	ge-0/0/1
			06:10.2	1	
			06:10.4	2	
			06:10.6	3	
06:00.1	560M	p2p2	06:10.1	0	
			06:10.3	1	
			06:10.5	2	
			06:10.7	3	
21:00.0	560FLB	em1	21:10.0	0	
			21:10.2	1	
			21:10.4	2	
			21:10.6	3	
21:00.1	560FLB	em2	21:10.1	0	
			21:10.3	1	
			21:10.5	2	
			21:10.7	3	
23:00.0	560M	p3p1	23:10.0	0	ge-0/0/0
			23:10.0	1	
			23:10.0	2	
			23:10.0	3	

PCI address	adapter	interface	VF address	VF#	VMX interface
23:00.1	560M	p3p2	23:10.1	0	
			23:10.3	1	
			23:10.5	2	
			23:10.7	3	

From this table, we seem to be able to conclude the following rules:

- a number of **10** represents "virtual" `device` (or `bus`)
- a sequence of "interleave" function numbers **0 2 4 6** represent the 4 "virtual" function **0** to **3** in the first physical port 0
- an "interleave" numbers **1 3 5 7** represent the 4 "virtual" function **0** to **3** in the second physical port 1

A question is : will this always hold true for other maximum VFs configuration? A quick test can show the answer.

17.6. example of 8 VFs:

As 1 VF and 4 VF examples shown above, to enable 8 VF per port (making it total 64 VFs platform wise), just change the `max_vfs` accordingly and use same commands will be sufficient:

```
sudo rmmod ixgbevf; \
sudo rmmod ixgbe; \
sudo modprobe ixgbe max_vfs=8,8,8,8,8,8,8,8; \
sudo modprobe tun ; \
sleep 5; sudo brctl addif br-ext em1
```

Now we can verify all the VF related data points using same commands as those used in previous examples.

to list all generated VFs

```
ping@trinity:/images/vmx_20151102.0/build$ lspci | grep -i ethernet
02:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
02:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
02:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
...<snippet>...
02:10.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
02:11.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
```

```
...<snippet>...
02:11.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
06:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
06:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
06:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
...<snippet>...
06:10.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
06:11.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
...<snippet>...
06:11.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
21:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
21:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
21:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
...<snippet>...
21:10.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
21:11.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
...<snippet>...
21:11.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
23:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
23:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
23:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
...<snippet>...
23:10.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
23:11.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
...<snippet>...
23:11.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
```

kernel message:

```
ping@trinity:/images/vmx_20151102.0/build$ dmesg | grep SR-IOV
[72130.750581] ixgbe 0000:02:00.0 (unregistered net_device): SR-IOV enabled with 8 VFs
[72130.986275] ixgbe 0000:02:00.1 (unregistered net_device): SR-IOV enabled with 8 VFs
[72131.223191] ixgbe 0000:06:00.0 (unregistered net_device): SR-IOV enabled with 8 VFs
[72131.458462] ixgbe 0000:06:00.1 (unregistered net_device): SR-IOV enabled with 8 VFs
[72131.693388] ixgbe 0000:21:00.0 (unregistered net_device): SR-IOV enabled with 8 VFs
[72131.937491] ixgbe 0000:21:00.1 (unregistered net_device): SR-IOV enabled with 8 VFs
[72132.173784] ixgbe 0000:23:00.0 (unregistered net_device): SR-IOV enabled with 8 VFs
[72132.409462] ixgbe 0000:23:00.1 (unregistered net_device): SR-IOV enabled with 8 VFs
```

PF-VF mapping:

```
ping@trinity:~$ ls -l /sys/bus/pci/devices/0000\:02\:00.0/virtfn*
lrwxrwxrwx 1 root root 0 Nov 24 14:10 /sys/bus/pci/devices/0000:02:00.0/virtfn0 ->
../0000:02:10.0
lrwxrwxrwx 1 root root 0 Nov 24 14:10 /sys/bus/pci/devices/0000:02:00.0/virtfn1 ->
../0000:02:10.2
lrwxrwxrwx 1 root root 0 Nov 24 14:10 /sys/bus/pci/devices/0000:02:00.0/virtfn2 ->
../0000:02:10.4
lrwxrwxrwx 1 root root 0 Nov 24 14:10 /sys/bus/pci/devices/0000:02:00.0/virtfn3 ->
../0000:02:10.6
lrwxrwxrwx 1 root root 0 Nov 24 14:10 /sys/bus/pci/devices/0000:02:00.0/virtfn4 ->
../0000:02:11.0
lrwxrwxrwx 1 root root 0 Nov 24 14:10 /sys/bus/pci/devices/0000:02:00.0/virtfn5 ->
../0000:02:11.2
lrwxrwxrwx 1 root root 0 Nov 24 14:10 /sys/bus/pci/devices/0000:02:00.0/virtfn6 ->
../0000:02:11.4
lrwxrwxrwx 1 root root 0 Nov 24 14:10 /sys/bus/pci/devices/0000:02:00.0/virtfn7 ->
../0000:02:11.6
```

17.6.1. the "rule of thumb" for 8 VF

Now we end up with below table for "8 VFs" scenario:

Table 8. NIC mapping table (8 VFs)

PCI address	adapter	interface	VF address	VF#	VMX interface
02:00.0	560FLB	em9	02:10.0	0	
			02:10.2	1	
			02:10.4	2	
			02:10.6	3	
			02:11.0	4	
			02:11.2	5	
			02:11.4	6	
			02:11.6	7	
02:00.1	560FLB	em10	02:10.1	0	
			02:10.3	1	
			02:10.5	2	
			02:10.7	3	
			02:11.1	4	
			02:11.3	5	
			02:11.5	6	
			02:11.7	7	
06:00.0	560M	p2p1	06:10.0	0	ge-0/0/1
			06:10.2	1	
			06:10.4	2	
			06:10.6	3	
			06:11.0	4	
			06:11.2	5	
			06:11.4	6	
			06:11.6	7	

PCI address	adapter	interface	VF address	VF#	VMX interface
06:00.1	560M	p2p2	06:10.1	0	
			06:10.3	1	
			06:10.5	2	
			06:10.7	3	
			06:11.1	4	
			06:11.3	5	
			06:11.5	6	
			06:11.7	7	
21:00.0	560FLB	em1	21:10.0	0	
			21:10.2	1	
			21:10.4	2	
			21:10.6	3	
			21:11.0	4	
			21:11.2	5	
			21:11.4	6	
			21:11.6	7	
21:00.1	560FLB	em2	21:10.1	0	
			21:10.3	1	
			21:10.5	2	
			21:10.7	3	
			21:11.1	4	
			21:11.3	5	
			21:11.5	6	
			21:11.7	7	

PCI address	adapter	interface	VF address	VF#	VMX interface
23:00.0	560M	p3p1	23:10.0	0	ge-0/0/0
			23:10.2	1	
			23:10.4	2	
			23:10.6	3	
			23:11.0	4	
			23:11.2	5	
			23:11.4	6	
			23:11.6	7	
23:00.1	560M	p3p2	23:10.1	0	
			23:10.3	1	
			23:10.5	2	
			23:10.7	3	
			23:11.1	4	
			23:11.3	5	
			23:11.5	6	
			23:11.7	7	

From this table, it looks the previous conclusions we got were almost right , except now we add one more device number **11**, so the rules is updated below:

- a number of **10** and **11** represents "virtual" `device` (or `bus`)
- a sequence of "interleave" function numbers **0 2 4 6** represent the 4 "virtual" function **0** to **3** under each virtual `device` number **10** and **11**, in the first physical port 0
- a sequence of "interleave" function numbers **1 3 5 7** represent the 4 "virtual" function **0** to **3** under each virtual virtual `device` number **10** and **11** in the second physical port 1

Chapter 18. virtio

18.1. virtio basic concept

comparing with "full virtualization" that QEMU software provided, `virtio` provides a "paravirtualization" environment, where guest operating system is "aware" that its running on a Hypervisor and cooperates with the hypervisor.

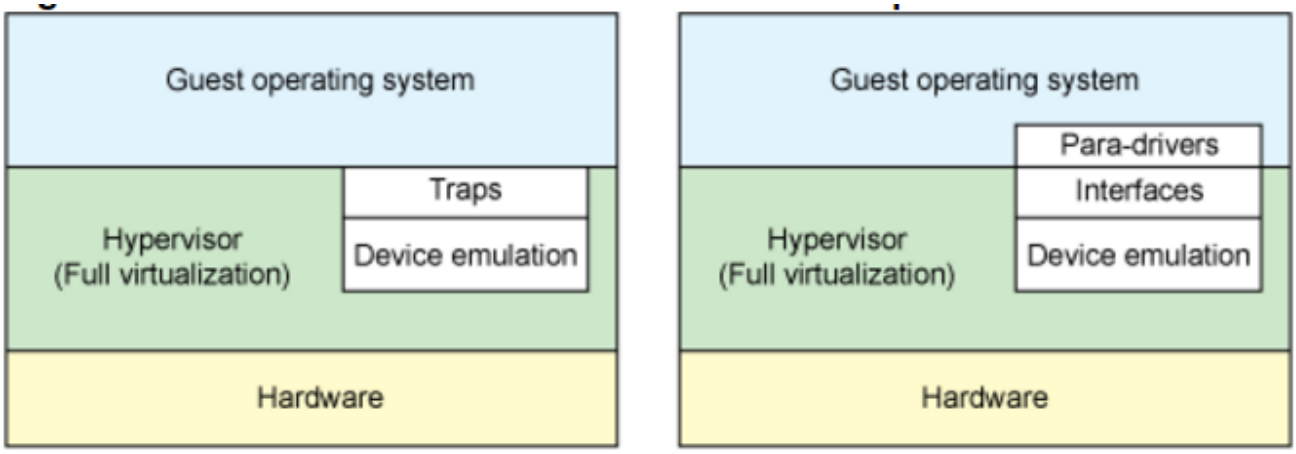


Figure 12. Device emulation in full virtualization and paravirtualization environments

`Virtio` is a virtualization standard for network and disk device drivers, which enables guests VM to get high performance network and disk operations, and gives most of the performance benefits of paravirtualization.

The key features of `virtio` are highlighted below:

- Virtio is an abstraction of common emulated devices like PCI, Hard drive and NICs
- guest OS needs to have virtio drivers, called "front end", to be able to work in this environment. examples of this front end drivers are:
  - `virtio-blk`
  - `virtio-net`
  - `virtio-balloon`.

currently `ubuntu` and most other linux variations have `virtio` driver supported in kernel hence no need any extra installations. for windows or other OSs, virtio drivers may need to be installed seperately.

- The hypervisor itself implements "backend drivers" for device emulation
- These "front end" and "backend drivers" work together to make up `virtio`
- Usually front end drivers are part of QEMU and back end drivers are part of KVM

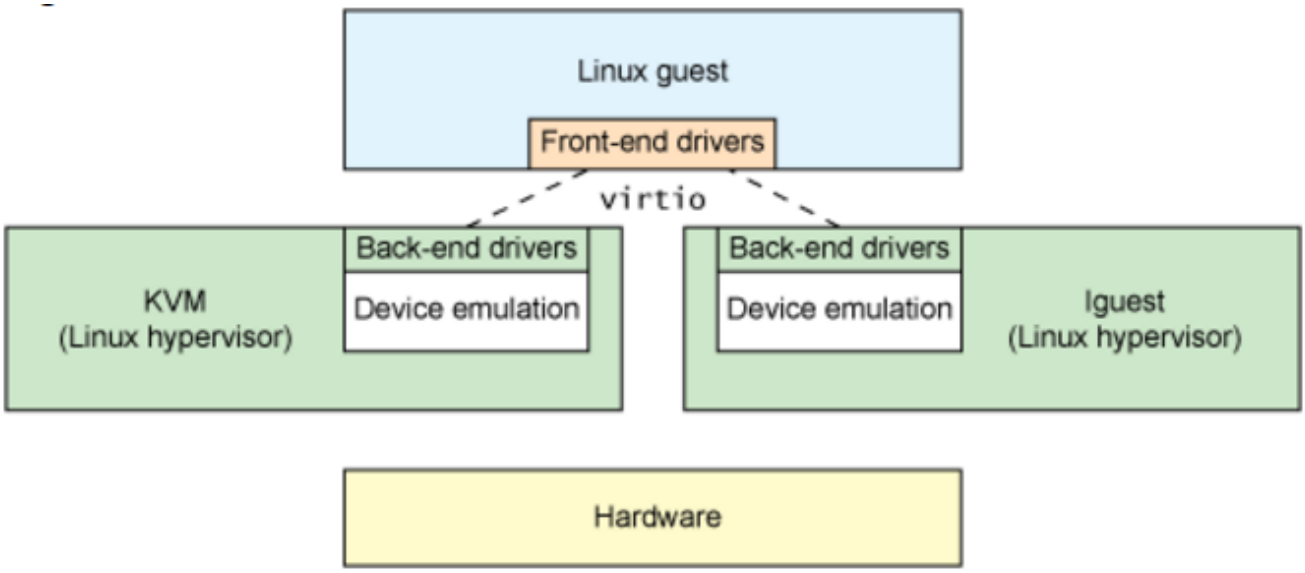


Figure 13. driver abstraction with virtio

In addition to the front-end drivers (implemented in the guest operating system) and the back-end drivers (implemented in the hypervisor), virtio defines two layers to support guest-to-hypervisor communication:

- At the top level (called virtio) is the "virtual queue" interface that conceptually attaches front-end drivers to back-end drivers.
- `virtio-ring` is used to buffer the info processed by the frontend/backend driver. it can also buffer the I/O requests from the frontend driver before hand over to the backend, improving the I/O processing efficiency.

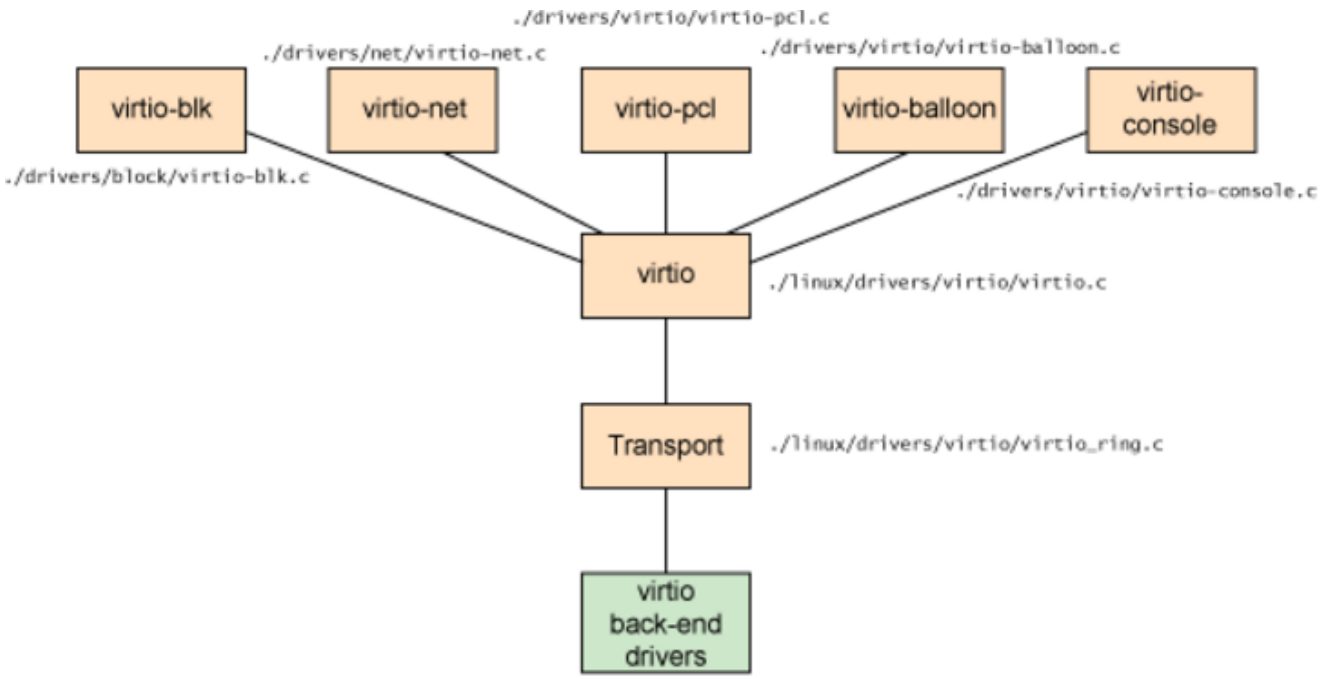


Figure 14. High-level architecture of the virtio framework

`virtio` can be used to create virtual NIC interfaces, independent of back end drivers in HyperVisor.

even For SR-IOV based MX86, virtio is still in use, but it is used only for vRE/vPFE internal and external management interfaces, not for data plane interfaces

virtio support in kvm

`Virtio` was chosen to be the main platform for IO virtualization in KVM. The idea behind it is to have a common framework for hypervisors for IO virtualization. At the moment, network/block/balloon devices are suported for kvm. The host implementation is in userspace - qemu, so no driver is needed in the host.

Virtio is relatively new technique and so it is not supported from first day of qemu-kvm, according to [linux-kvm-virtio], kvm version needs to be `>60` and linux kernel needs to be later than `2.6.25`. Also it needs some specific configuration options to be activated in the kernel. To verify if the QEMU-KVM installed in the server support virtio, run this command:

```
ping@trinity:~$ qemu-system-x86_64 -net nic,model=?
qemu: Supported NIC models:
ne2k_pci,i82551,i82557b,i82559er,rtl8139,e1000,pcnet,virtio

^^^^^^ ①
```

① virtio is supported in this qemu-kvm release.

virtio in VMX

the installation of virtio version of VMX does not enforce the following requirements:

- physical NIC with SR-IOV capability
- IXGBE driver
- specific linux kernel version (e.g. 3.13)
- IOMMU/VT-d

Therefore if performance is not a concern, it's more convenient to setup `virtio` version of VMX for learning/testing purpose.

virtio linux kernel driver support

Current linux kernel already has `virtio` driver module support:

```
ping@trinity:~$ grep -i virtio /boot/config-3.13.0-32-generic
CONFIG_NET_9P_VIRTIO=m
CONFIG_VIRTIO_BLK=y
CONFIG_SCSI_VIRTIO=m
CONFIG_VIRTIO_NET=y
CONFIG_CAIF_VIRTIO=m
CONFIG_VIRTIO_CONSOLE=y
CONFIG_HW_RANDOM_VIRTIO=m
CONFIG_VIRTIO=y
# Virtio drivers
CONFIG_VIRTIO_PCI=y
CONFIG_VIRTIO_BALLOON=y
CONFIG_VIRTIO_MMIO=y
CONFIG_VIRTIO_MMIO_CMDLINE_DEVICES=y

ping@trinity:~$ find /lib/modules/3.13.0-32-generic/ -name "virtio*"
/lib/modules/3.13.0-32-generic/kernel/drivers/scsi/virtio_scsi.ko
/lib/modules/3.13.0-32-generic/kernel/drivers/char/hw_random/virtio-rng.ko
```

18.2. virtio verification in vPFE guest VM

logging into vPFE VM and more details about virtio can be observed.

1. loggin into vPFE

   ```
   ping@trinity:~$ telnet localhost 8817
   Trying ::1...
   Trying 127.0.0.1...
   Connected to localhost.
   Escape character is '^]'.
   Wind River Linux 6.0.0.12 vfp-vmx1 console
   vfp-vmx1 login: pfe
   Password:
   ```

2. para-virtualization

   `para-virtualization` simply means guest VM is "aware of" the virtualization environment, which can be verified by looking at the vFPC guest VM booting log:

```
/boot/modules/virtio.ko size 0x69a0 at 0x1616000
/boot/modules/virtio_pci.ko size 0x6fb8 at 0x161d000
/boot/modules/virtio_blk.ko size 0x7988 at 0x1624000
/boot/modules/if_vtnet.ko size 0x34f10 at 0x162c000
/boot/modules/virtio_console.ko size 0x9740 at 0x1661000

virtio_pci0: <VirtIO PCI Network adapter> port 0xc560-0xc57f mem
    0xfebf1000-0xfebf1fff irq 10 at device 5.0 on pci0
em1: <VirtIO Networking Adapter> on virtio_pci0
virtio_pci0: host features: 0x511fffe3
    <RingIndirect,NotifyOnEmpty,RxModeExtra,VLanFilter,RxMode,ControlVq,Status,
    MrgRxBuf,TxUFO,TxTSOECN,TxTSOv6,TxTSOv4,RxUFO,RxECN,RxTSOv6,RxTSOv4,TxAllGSO,
    MacAddress,RxChecksum,TxChecksum>
virtio_pci0: negotiated features: 0x110f8020 <RingIndirect,NotifyOnEmpty,
    VLanFilter,RxMode,ControlVq,Status,MrgRxBuf,MacAddress>
virtio_pci1: <VirtIO PCI Balloon adapter> port 0xc580-0xc59f irq 10 at device
    6.0 on pci0
```

3. virtio NIC

   To view the emulated PCI devices, use the same `lspci` command in the vPFE guest VM, as what we did in the host:

```
pfe@vfp-vmx1:~$ lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.2 USB controller: Intel Corporation 82371SB PIIX3 USB [Natoma/Triton II] (rev
01)
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Cirrus Logic GD 5446
00:03.0 Ethernet controller: Red Hat, Inc Virtio network device         ①
00:04.0 Ethernet controller: Red Hat, Inc Virtio network device         ②
00:05.0 Ethernet controller: Red Hat, Inc Virtio network device         ③
00:06.0 Ethernet controller: Red Hat, Inc Virtio network device         ④
00:07.0 Multimedia audio controller: Intel Corporation 82801AA AC'97 Audio
Controller (rev 01)
00:08.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
```

   The output captured here matches to the `virsh` command that we demonstrated previously, listing all VM virtual network(VN) info using KVM qmp command `info network` from outside of the VM (from host):

```
ping@trinity:~$ sudo virsh qemu-monitor-command vcp-vmx1 --hmp "info network"
[sudo] password for ping:
net0: index=0,type=nic,model=e1000,macaddr=02:04:17:01:01:01
 \ hostnet0: index=0,type=tap,fd=18
net1: index=0,type=nic,model=virtio-net-pci,macaddr=52:54:00:10:91:fe
 \ hostnet1: index=0,type=tap,fd=19

ping@trinity:~$ sudo virsh qemu-monitor-command vfp-vmx1 --hmp "info network"
net0: index=0,type=nic,model=virtio-net-pci,macaddr=02:04:17:01:01:02   ①
 \ hostnet0: index=0,type=tap,fd=18
net1: index=0,type=nic,model=virtio-net-pci,macaddr=52:54:00:db:34:a9   ②
 \ hostnet1: index=0,type=tap,fd=21
net2: index=0,type=nic,model=virtio-net-pci,macaddr=02:04:17:01:02:01   ③
 \ hostnet2: index=0,type=tap,fd=23
net3: index=0,type=nic,model=virtio-net-pci,macaddr=02:04:17:01:02:02   ④
 \ hostnet3: index=0,type=tap,fd=25
```

① virtio interface for vPFE external mgmt interface: `ext`

② virtio interface for vPFE internal mgmt interface: `int`

③ virtio interface for JUNOS interface `ge-0/0/0`

④ virtio interface for JUNOS interface `ge-0/0/1`

to get more detail about the virtio emulated PCI device:

```
pfe@vfp-vmx1:~$ lspci -vvks 00:03.0
00:03.0 Ethernet controller: Red Hat, Inc Virtio network device
        Subsystem: Red Hat, Inc Device 0001
        Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr-
Stepping- SERR+ FastB2B- DisINTx+
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort-
<MAbort- >SERR- <PERR- INTx-
        Latency: 0
        Interrupt: pin A routed to IRQ 10
        Region 0: I/O ports at c520 [size=32]
        Region 1: Memory at febd1000 (32-bit, non-prefetchable) [size=4K]
        Expansion ROM at feac0000 [disabled] [size=256K]
        Capabilities: <access denied>
        Kernel driver in use: virtio-pci        #<------
```

as expected, currently `virtio` driver module is in use on this virtio emulated virtual NICs.

4. virtio-balloon

   besides NIC, virtio also brings a memory-saving technique called "balloon", which will be covered in more details later.

```
pfe@vfp-vmx1:~$ lspci -vvks 00:08.0
00:08.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
        Subsystem: Red Hat, Inc Device 0005
        Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr-
Stepping- SERR+ FastB2B- DisINTx-
        Status: Cap- 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort-
<MAbort- >SERR- <PERR- INTx-
        Latency: 0
        Interrupt: pin A routed to IRQ 11
        Region 0: I/O ports at c5a0 [size=32]
        Kernel driver in use: virtio-pci
```

TODO

Chapter 19. cpu pinning (affinitization)

19.1. 2 terms

There are 2 related terms regarding CPU pinning:

CPU pinning

> CPU pinning is the ability to run specific VM's virtual CPU (vCPU) on specific physical CPU (pCPU) in a specific host.

As explained in vcpu essential, a "vcpu" essentially is nothing but a thread running inside the space of a QEMU process. Therefore, pinning a "vCPU" is pinning a thread - restricting a thread to run on a dedicated physical CPU.

Restricting a thread to run on a single CPU avoids the performance cost caused by the cache invalidation that occurs when a thread ceases to execute on one CPU and then recommences execution on a different CPU. this is why CPU pinning usually gain performance - if tuned properly.

CPU affinity

> CPU affinity is a scheduler property that "bonds" a process to a given set of CPUs on the system. The Linux scheduler will honor the given CPU affinity and the process will not run on any other CPUs. The CPU affinity is represented as a bitmask, with the lowest order bit corresponding to the first logical CPU and the highest order bit corresponding to the last logical CPU. The masks are typically given in hexadecimal.

A thread's `CPU affinity mask` determines the set of CPUs on which it is eligible to run. On a multiprocessor system, vCPU pinning can be internally archived by setting the `CPU affinity mask`.

It is possible to ensure maximum execution speed for that thread, by dedicating one CPU to a particular thread:

- setting the affinity mask of that thread to specify a single CPU
- setting the affinity mask of all other threads to exclude that CPU)

for example:

affinity mask	eligibile processors
0x00000001	processor #0
0x00000003	processors #0 and #1
0xFFFFFFFF	all processors (#0 through #31).

the Linux scheduler also supports natural CPU affinity: the scheduler attempts to keep pro- cesses on the same CPU as long as practical for performance reasons. Therefore, forcing a specific CPU affinity is useful only in certain applications.

Not all CPUs may exist on a given system but a mask may specify more CPUs than are present. A retrieved mask will reflect only the bits that correspond to CPUs physically on the system.

If an invalid mask is given (i.e., one that corresponds to no valid CPUs on the current system) an error is returned.

19.2. 2 `virsh` commands

There are at least 2 commands in `libvirt` `virsh` tool regarding "CPU affinity" feature:

`vcpuinfo`
: used to retrieve the CPU affinity of a running VM.

`vcpupin`
: used to "pin" guest domain virtual CPUs to physical host CPUs.

these 2 `virsh` commands play a very similiar roll as what linux utility `taskset` does, which is used to set or retrieve the CPU affinity of a running process given its PID or to launch a new COMMAND with a given CPU affinity.

19.3. a simple test

here is a simple test to demonstrate this feature:

first, list vcpuinfo BEFORE vcpupin:

vmx1 vRE:

```
ping@trinity:~$ sudo virsh vcpuinfo vcp-vmx1
VCPU:           0
CPU:            0
State:          running
CPU time:       93.2s
CPU Affinity:   y---------------------------- ①
```

vmx1 vFPC:

```
ping@trinity:~$ sudo virsh vcpuinfo vfp-vmx1
VCPU:           0
CPU:            6
State:          running
CPU time:       18.1s
CPU Affinity:   yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy ①

VCPU:           1
CPU:            24
State:          running
CPU time:       12.9s
CPU Affinity:   yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy ①

VCPU:           2
CPU:            28
State:          running
CPU time:       12.4s
CPU Affinity:   yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy ①

VCPU:           3
CPU:            5
State:          running
CPU time:       12.2s
CPU Affinity:   yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy ①
```

① "CPU Affinity mask". a `y` on a bit indicate the eligibility of the specific CPU NO. to run current process.

as can be seen before CPU pinning, the 4 VCPU/threads can be running in any of the physical cores, because the "CPU Affinity" is all `y`.

This is because by default, `libvirt` provisions guests using the hypervisor's default policy. For most hypervisors, the policy is to run guests on any available processing core or CPU.

There are times when an explicit policy may be better, in particular for systems with a NUMA (Non-Uniform Memory Access) architecture. A guest on a NUMA system should be pinned to a processing core so that its memory allocations are always local to the node it is running on. This avoids cross-node memory transports which have less bandwidth and can significantly degrade performance.

```
virsh vcpupin vfp-vmx1 0 11
virsh vcpupin vfp-vmx1 1 12
virsh vcpupin vfp-vmx1 2 13
virsh vcpupin vfp-vmx1 3 8
virsh vcpupin vfp-vmx1 4 9
virsh vcpupin vcp-vmx1 0 15
virsh emulatorpin vcp-vmx1 0
virsh emulatorpin vfp-vmx1 0
```

This is the current mapping between vCPU(thread) and CPU (to shortly the capture we ignored the `cpuinfo` output from vmx2 instance):

vCPU(vmx1)	CPU	vCPU(vmx2)	CPU	Affinity
0(vcp)	0	0(vcp)	0	any CPU
0	6	0	27	any CPU
1	24	1	31	any CPU
2	28	2	25	any CPU
3	5	3	30	any CPU

after vcpupin

```
ping@trinity:~$ sudo virsh vcpuinfo vcp-vmx1
VCPU:           0
CPU:            7
State:          running
CPU time:       465.7s
CPU Affinity:   -------y------------------------

ping@trinity:~$ sudo virsh vcpuinfo vfp-vmx1
VCPU:           0
CPU:            0
State:          running
CPU time:       571.7s
CPU Affinity:   y-------------------------------

VCPU:           1
CPU:            1
State:          running
CPU time:       1432.5s
CPU Affinity:   -y------------------------------

VCPU:           2
CPU:            2
State:          running
CPU time:       1133.6s
CPU Affinity:   --y-----------------------------

VCPU:           3
CPU:            3
State:          running
CPU time:       517.1s
CPU Affinity:   ---y----------------------------
```

vCPU(vmx1)	CPU	vCPU(vmx2)	CPU	Affinity
0(vcp)	7	0(vcp)	15	only specified CPU
0	0	0	8	only specified CPU
1	1	1	9	only specified CPU
2	2	2	10	only specified CPU
3	3	3	11	only specified CPU

Chapter 20. hugepage

20.1. hugepage basic concept

Whenever a process uses some memory, CPU will mark the RAM as used by that process. For efficiency, x86 CPU allocate RAM by chunks of 4K bytes, which is named as "a page". Those pages can be swapped to disk, etc.

Since the process address space are virtual, the CPU and the operating system have to remember which page belong to which process, and where it is stored. the data structure that is used to maintain this info is called `TLB` (translation lookaside buffer) , which is often implemented as `CAM` (content-addressable memory).

The CAM search key is the virtual address and the search result is a physical address. If the requested address is present in the TLB, the CAM search yields a match quickly and the retrieved physical address can be used to access memory. This is called a `TLB hit`. If the requested address is not in the TLB, it is a `cache miss`.

Obviously, the more pages you have, the more time it takes the CPU to search the TLB for the target memory. Increasing page size (hence called `Hugepage`) will reduce the amount of pages, and eventually improving the CPU performance.

Most current CPU architectures support hugepage, but possibly with a different name/term, but they are all the same thing:

- Huge pages (on Linux)
- Super Pages (on BSD)
- Large Pages (on Windows)

20.2. hugepage allocation

The allocation of hugepages should be done at boot time or as soon as possible after system boot to prevent memory from being fragmented in physical memory. To reserve `hugepages` at boot time, a parameter is passed to the Linux* kernel on the kernel command line.

x86 by default use 4K page size

```
ping@ubuntu1:~$ getconf PAGESIZE
4096
```

```
root@ubuntu1:~# getconf -a | grep -i page
PAGESIZE                           4096
PAGE_SIZE                          4096
_AVPHYS_PAGES                      72309823
_PHYS_PAGES                        99076139
```

VMX use hugepage size **2M**, and total number of hugepages is configured to be 32k, make it 64G hugepage memory for the guest VMs.

before hugepage was enabled:

```
ping@trinity:/images/vmx_20151102.0/build$ cat /proc/meminfo | grep -i huge
AnonHugePages:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

after:

```
root@ubuntu1:~# grep -i page /proc/meminfo
AnonPages:       2244644 kB
PageTables:        17280 kB
AnonHugePages:   2076672 kB
HugePages_Total:   32786            #<------
HugePages_Free:    24969
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

same info can be printed from kernel options using sysctl utility:

```
ping@ubuntu1:/sys/module/ixgbevf$ sudo sysctl -a | grep -i hugepage
vm.hugepages_treat_as_movable = 0
vm.nr_hugepages = 32786     #<------
vm.nr_hugepages_mempolicy = 32786
vm.nr_overcommit_hugepages = 0
```

```
ping@ubuntu1:~$ sysctl vm.nr_hugepages
vm.nr_hugepages = 32786
```

In ubuntu system, and all the other linux variants, these parameters are stored in the below file systems:

- under /proc/sys/vm/ folder in the /proc file system
- under /sys/kernel/mm/hugepages/ folder in the /sys file system

```
ping@ubuntu1:~$ grep -R "" /proc/sys/vm/*huge*
/proc/sys/vm/hugepages_treat_as_movable:0
/proc/sys/vm/hugetlb_shm_group:0
/proc/sys/vm/nr_hugepages:32768
/proc/sys/vm/nr_hugepages_mempolicy:32768
/proc/sys/vm/nr_overcommit_hugepages:0
```

```
ping@ubuntu1:~$ grep -R "" /sys/kernel/mm/hugepages/
/sys/kernel/mm/hugepages/hugepages-2048kB/nr_overcommit_hugepages:0
/sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages:32768
/sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages_mempolicy:32768
/sys/kernel/mm/hugepages/hugepages-2048kB/surplus_hugepages:0
/sys/kernel/mm/hugepages/hugepages-2048kB/resv_hugepages:0
/sys/kernel/mm/hugepages/hugepages-2048kB/free_hugepages:24951
ping@ubuntu1:~$
```

about `/proc` and `/sys`

`/proc` is very special in that it is also a virtual filesystem. It's sometimes referred to as a process information pseudo-file system. It doesn't contain 'real' files but runtime system information (e.g. system memory, devices mounted, hardware configuration, etc). For this reason it can be regarded as a control and information centre for the kernel. In fact, quite a lot of system utilities are simply calls to files in this directory. For example, 'lsmod' is the same as 'cat /proc/modules' while 'lspci' is a synonym for 'cat /proc/pci'. By altering files located in this directory you can even read/change kernel parameters (sysctl) while the system is running. refer to [proc_file_system] for more details of proc file system.

`sysfs` is a feature of the Linux 2.6 kernel that allows kernel code to export information to user processes via an in-memory filesystem. The organization of the filesystem directory hierarchy is strict, and based the internal organization of kernel data structures. The files that are created in the filesystem are (mostly) ASCII files with (usually) one value per file. These features ensure that the information exported is accurate and easily accessible, making sysfs one of the most intuitive and useful features of the 2.6 kernel.

Another tool to query the same data is `hugeadm`:

```
ping@ubuntu1:~$ hugeadm --pool-list
      Size  Minimum  Current  Maximum  Default
   2097152    32768    32768    32768        *
```

Interestingly, with `--explain` option it shows some "explanation" about the output.

```
ping@ubuntu1:~$ hugeadm --pool-list --explain
      Size  Minimum  Current  Maximum  Default
   2097152    32768    32768    32768        *
Total System Memory: 387016 MB
```

```
Mount Point          Options
/run/hugepages/kvm   rw,relatime,mode=775,gid=112
/HugePage_vPFE       rw,relatime
/HugePage_vPFE       rw,relatime
```

```
Huge page pools:
      Size  Minimum  Current  Maximum  Default
   2097152    32768    32768    32768        *
```

```
Huge page sizes with configured pools:
```

```
A /proc/sys/kernel/shmmax value of 33554432 bytes may be sub-optimal. To maximise
shared memory usage, this should be set to the size of the largest shared memory
segment size you want to be able to use. Alternatively, set it to a size matching
the maximum possible allocation size of all huge pages. This can be done
automatically, using the --set-recommended-shmmax option.
```

```
The recommended shmmax for your currently allocated huge pages is 68719476736 bytes.
To make shmmax settings persistent, add the following line to /etc/sysctl.conf:
  kernel.shmmax = 68719476736
```

```
hugeadm:WARNING: User ping (uid: 1003) is not a member of the hugetlb_shm_group root
(gid: 0)!
```

```
Note: Permanent swap space should be preferred when dynamic huge page pools are used.
```

20.3. change hugepages number

To change the number of hugepages in realtime, use kernel tool `sysctl`. in this example we change hugepages from `32786` to `32768`:

```
ping@ubuntu1:~$ sudo sysctl vm.nr_hugepages=32768
vm.nr_hugepages = 32768
```

Now the number of total hugepages are changed, as well as the free hugepages:

```
ping@ubuntu1:~$ grep -i huge /proc/meminfo
AnonHugePages:   2076672 kB
HugePages_Total:   32768        #<------
HugePages_Free:    24951        #<------
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

20.4. hugepage and numa

The huage pages are distributed between all numanode, in this server there are 2 numanode , so each of node were allocated 16384 pages. the interesting part is the number of free hugepages in each node: the hugepages from first node `0` will be consumed first (by VMX instance in our context), so only `8567` pages are left. VMX consumed `7817` pages, which is `15634M` memory.

```
ping@ubuntu1:~$ cat /sys/devices/system/node/node*/meminfo | grep Huge
Node 0 AnonHugePages:     88064 kB
Node 0 HugePages_Total: 16384   #<------
Node 0 HugePages_Free:   8567
Node 0 HugePages_Surp:      0
Node 1 AnonHugePages:   2050048 kB
Node 1 HugePages_Total: 16384   #<------
Node 1 HugePages_Free:  16384
Node 1 HugePages_Surp:      0
```

The hugepages from the second node (node1) will be consumed if we start another VMX instance. The hugepage allocation were performed by `libvirt` "under the scene".

20.5. a simple test of hugepage

currently `24951` hugepages are free:

```
ping@ubuntu1:~$ grep -i huge /proc/meminfo
AnonHugePages:   2076672 kB
HugePages_Total:   32768
HugePages_Free:    24951        #<------
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

if we destroy VPFE VM:

```
ping@ubuntu1:~$ sudo virsh destroy 36
Domain 36 destroyed
```

now all hugepages were claimed back:

```
ping@ubuntu1:~$ grep -i huge /proc/meminfo
AnonHugePages:   2068480 kB
HugePages_Total:   32768
HugePages_Free:    32768        #<------
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

start VM:

```
ping@ubuntu1:~$ sudo virsh start vhepe-vfp
Domain vhepe-vfp started
```

free huagepages were again consumed by the VM.

```
ping@ubuntu1:~$ grep -i huge /proc/meminfo
AnonHugePages:   2076672 kB
HugePages_Total:   32768
HugePages_Free:    24951        #<------
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

totally `(32768-24951) X 2048K = 15634M` hugepage memory were allocated to the vPFE VM.

refer to huge page for more details on this topic.

appendix

reference

VMX official document

- VMX official document/software download
- vmx 15.1F3 release note

intel spec

- x86 virtualization
- HP Ethernet 10Gb 2-port 560FLB Adapter
- Intel 82599 10GE controller
- E5-4627 spec

libvirt/virsh

- libvirt FAQ
- libvirt network XML format
- Virsh Command Reference
- libvirt 1.2.8 release
- libvirt virtual networking

virtio

- libvirt virtio
- Virtio: An I/O virtualization framework for Linux
- [linux-kvm-virtio] virtio
- virtio: Towards a De-Facto Standard For Virtual I/O Devices
- virtio balloon

VT-d/PCI passthrough

- How to assign devices with VT-d in KVM
- PCI passthrough with virsh
- linux PCI passthough
- VT-d spec
- Intel VT-d

SR-IOV

- Implementing SR-IOV for Linux on HP ProLiant servers
- PCI-SIG SR-IOV Primer
- SR-IOV white paper
- SR-IOV

KVM/qemu features

- huge page
- qemu changelog
- qmp
- LIBVIRT NUMA TUNING
- Setting KVM processor affinities

DPDK

- dpdk-getting-started-guide
- dpdk supported nics

misc

- BDF notation
- proc file system
- [sys_file_system] sys file system
- posix thread programming

VMX package folder structure

```
ping@trinity:~/vmx_20151102.0$ tree
.
├──── config                          #<------all config files that will be read
│     ├──── samples                             and parsed by the installation script
│     │     ├──── vmx.conf.sriov
│     │     ├──── vmx.conf.virtio
│     │     └──── vmx-galaxy.conf
│     ├──── vmx.conf                    #<------the main config file used
│     └──── vmx-junosdev.conf
├──── docs
│     ├──── mx86_on_openstack_release_notes.pdf
│     └──── VMX_Release_Notes_Installation_Guide_Beta.pdf
├──── drivers
│     ├──── galaxy
│     │     ├──── Makefile
│     │     └──── network_add.c
│     └──── ixgbe-3.19.1                #<------ixgbe driver source code
│         ├──── COPYING
│         ├──── ixgbe.7
│         ├──── ixgbe.spec
│         ├──── pci.updates
│         ├──── README
│         ├──── scripts
│         │     └──── set_irq_affinity
│         ├──── src
│         │     ├──── ixgbe.7.gz
│         │     ├──── ixgbe_82598.c
│         │     ├──── ixgbe_82598.h
│         │     ├──── ixgbe_82599.c
│         │     ├──── ixgbe_82599.h
│         │     ├──── ixgbe_api.c
│         │     ├──── ixgbe_api.h
│         │     ├──── ixgbe_common.c
│         │     ├──── ixgbe_common.h
│         │     ├──── ixgbe_dcb_82598.c
│         │     ├──── ixgbe_dcb_82598.h
│         │     ├──── ixgbe_dcb_82599.c
│         │     ├──── ixgbe_dcb_82599.h
│         │     ├──── ixgbe_dcb.c
│         │     ├──── ixgbe_dcb.h
│         │     ├──── ixgbe_dcb_nl.c
│         │     ├──── ixgbe_debugfs.c
│         │     ├──── ixgbe_ethtool.c
│         │     ├──── ixgbe_fcoe.c
│         │     ├──── ixgbe_fcoe.h
│         │     ├──── ixgbe.h
│         │     ├──── ixgbe_lib.c
```

```
│         │     ├──── ixgbe_main.c
│         │     ├──── ixgbe_mbx.c
│         │     ├──── ixgbe_mbx.h
│         │     ├──── ixgbe.mod.c
│         │     ├──── ixgbe_osdep2.h
│         │     ├──── ixgbe_osdep.h
│         │     ├──── ixgbe_param.c
│         │     ├──── ixgbe_phy.c
│         │     ├──── ixgbe_phy.h
│         │     ├──── ixgbe_procfs.c
│         │     ├──── ixgbe_ptp.c
│         │     ├──── ixgbe_sriov.c
│         │     ├──── ixgbe_sriov.h
│         │     ├──── ixgbe_sysfs.c
│         │     ├──── ixgbe_type.h
│         │     ├──── ixgbe_x540.c
│         │     ├──── ixgbe_x540.h
│         │     ├──── kcompat.c
│         │     ├──── kcompat_ethtool.c
│         │     ├──── kcompat.h
│         │     ├──── Makefile
│         │     ├──── modules.order
│         │     ├──── Module.supported
│         │     └──── Module.symvers
│         └──── SUMS
├──── env
│     ├──── ubuntu_sriov.env
│     └──── ubuntu_virtio.env
├──── images                          #<------all VM images that will be
│     ├──── metadata_usb.img                    loaded by the KVM-qumu hypervisor
│     ├──── vFPC-20151102.img
│     └──── vmxhdd.img
├──── scripts                         #<------installation scripts
│     ├──── common
│     │     ├──── get_numa.sh
│     │     ├──── get_pci.sh
│     │     ├──── vmx_common_utils.sh
│     │     ├──── vmx_configure.py        #<------python script to parse YAML config
file
│     │     ├──── vmx_env.sh                      and generate the libvirt friendly XML
files
│     │     ├──── vmx_galaxy.py
│     │     ├──── vmx_img_cli.sh
│     │     └──── vmx_preinstall_checks.sh
│     ├──── junosdev-bind
│     │     ├──── vmx_brctl.sh
│     │     ├──── vmx-junosdev-bind.py
│     │     ├──── vmx_linkctl.sh
│     │     └──── vmx_vhost_pin.sh
│     ├──── kvm
│     │     ├──── common
```

```
│   │   │   ├── vmx_kvm_bringup.sh
│   │   │   ├── vmx_kvm_cleanup.sh
│   │   │   ├── vmx_kvm_system_setup.sh
│   │   │   └── vmx_kvm_verify.sh
│   │   ├── sriov
│   │   │   └── vmx_kvm_sriov.sh
│   │   └── virtio
│   │       └── vmx_kvm_virtio.sh
│   └── templates
│       ├── _br_ext-ref.xml
│       ├── _br_int-ref.xml
│       ├── _vPFE-ref-ubuntu.xml
│       ├── _vPFE-ref.xml
│       └── _vRE-ref.xml
└── vmx.sh                          #<------the main script to be executed

18 directories, 91 files
```

VMX installation script generated XML file

generated vRE xml

build/vmx1/xml/vRE-generated.xml:

```
ping@trinity:~$ cat xml.backup/vRE-generated.xml
<domain type="kvm">

  <name>vcp-vmx1</name>                                #<------domain name (VM)

  <memory unit="Mb">2048</memory>                      #<------memory: 2G

  <vcpu placement="static">1</vcpu>                    #<------1 CPU for RE

  <cputune>                                            #<------finetune: vcpupin
    <vcpupin cpuset="0" vcpu="0" />
  </cputune>

  <resource>
    <partition>/machine</partition>
  </resource>

  <sysinfo type="smbios">
    <bios>
      <entry name="vendor">Juniper</entry>
    </bios>
    <system>
      <entry name="manufacturer">VMX</entry>
      <entry name="product">VM-vcp_vmx1-161-re-0</entry>
      <entry name="version">0.1.0</entry>
    </system>
  </sysinfo>

  <os>                                                 #<------os info
    <smbios mode="sysinfo" />
    <type arch="x86_64" machine="pc-0.13">hvm</type>
    <boot dev="hd" />                                  #<------boot sequence
  </os>

  <features>                                           #<------HW featured supported
    <acpi />
    <apic />
    <pae />
  </features>

  <cpu mode="host-model">
    <topology cores="1" sockets="1" threads="1" />
  </cpu>
```

```
  <clock offset="utc" />

  <on_poweroff>destroy</on_poweroff>

  <on_reboot>restart</on_reboot>

  <on_crash>restart</on_crash>

  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>

    <disk device="disk" type="file">
      <driver cache="directsync" name="qemu" type="qcow2" />
      <source file="/images/vmx_20151102.0/build/vmx1/images/jinstall64-vmx-15.1F-
20151104.0-domestic.img" />
      <target bus="ide" dev="hda" />
      <address bus="0" controller="0" target="0" type="drive" unit="0" />
    </disk>

    <disk device="disk" type="file">
      <driver cache="directsync" name="qemu" type="qcow2" />
      <source file="/images/vmx_20151102.0/build/vmx1/images/vmxhdd.img" />
      <target bus="ide" dev="hdb" />
      <address bus="0" controller="0" target="0" type="drive" unit="1" />
    </disk>

    <disk device="disk" type="file">
      <driver cache="directsync" name="qemu" type="raw" />
      <source file="/images/vmx_20151102.0/images/metadata_usb.img" />
      <target bus="usb" dev="sda" />
    </disk>

    <controller index="0" type="usb">               #<------pci device
      <address bus="0x00" domain="0x0000" function="0x2" slot="0x01" type="pci" />
    </controller>
    <controller index="0" type="ide">
      <address bus="0x00" domain="0x0000" function="0x1" slot="0x01" type="pci" />
    </controller>
    <controller index="0" model="pci-root" type="pci" />

    <interface type="bridge">               #<------generate bridge interface
      <source bridge="br-ext" />            #<------named vcp_ext-vmx1
      <target dev="vcp_ext-vmx1" />         #<------bound to br-ext bridge
      <model type="e1000" />                #<------soft emulation
      <mac address="02:12:DE:C0:DE:22" />
    </interface>

    <interface type="bridge">               #<------internal bridge
      <source bridge="br-int-vmx1" />
      <target dev="vcp_int-vmx1" />
```

```
      <model type="virtio" />              #<------virtio emulation
    </interface>

    <serial type="tcp">
      <source host="127.0.0.1" mode="bind" service="8896" />
      <protocol type="telnet" />
      <target port="0" />
    </serial>

    <console type="tcp">
      <source host="127.0.0.1" mode="bind" service="8896" />
      <protocol type="telnet" />
      <target port="0" type="serial" />
    </console>

    <input bus="usb" type="tablet" />
    <input bus="ps2" type="mouse" />
    <input bus="ps2" type="keyboard" />

    <graphics autoport="yes" listen="127.0.0.1" port="-1" type="vnc">
                                          #<------automatic vnc server port
      <listen address="127.0.0.1" type="address" />
    </graphics>

    <sound model="ac97">
      <address bus="0x00" domain="0x0000" function="0x0" slot="0x04" type="pci" />
    </sound>

    <video>
      <model heads="1" type="cirrus" vram="9216" />
      <address bus="0x00" domain="0x0000" function="0x0" slot="0x02" type="pci" />
    </video>

    <memballoon model="virtio">            #<------memory balloon location
      <address bus="0x00" domain="0x0000" function="0x0" slot="0x06" type="pci" />
    </memballoon>

  </devices>
</domain>
```

generated vPFE xml

build/vmx1/xml/vPFE-generated.xml:

```
ping@trinity:~$ cat xml.backup/vPFE-generated.xml
<domain type="kvm">

  <name>vfp-vmx1</name>
```

```
<memory unit="MB">16384</memory>

<memoryBacking>
  <nosharepages />
  <hugepages />
</memoryBacking>

<vcpu placement="static">4</vcpu>

<numatune>
  <memory mode="strict" nodeset="1" />
</numatune>

<os>
  <type arch="x86_64" machine="pc-i440fx-trusty">hvm</type>
  <boot dev="hd" />
</os>

<features>
  <acpi />
</features>

<cpu mode="host-model">
  <topology cores="4" sockets="1" threads="1" />
</cpu>

<clock offset="utc" />
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>

<devices>
  <emulator>/usr/bin/qemu-system-x86_64</emulator>

  <disk device="disk" type="file">
    <driver cache="directsync" name="qemu" type="raw" />
    <source file="/images/vmx_20151102.0/build/vmx1/images/vFPC-20151102.img" />
    <target bus="ide" dev="hda" />
  </disk>

  <controller index="0" model="pci-root" type="pci" />

  <interface type="bridge">
    <source bridge="br-ext" />
    <target dev="vfp_ext-vmx1" />
    <model type="virtio" />
    <alias name="net0" />
    <address bus="0x00" domain="0x0000" function="0x0" slot="0x03" type="pci" />
    <mac address="02:12:DE:C0:DE:23" />
  </interface>
```

```
    <interface type="bridge">
      <source bridge="br-int-vmx1" />
      <target dev="vfp_int-vmx1" />
      <model type="virtio" />
      <alias name="net0" />
      <address bus="0x00" domain="0x0000" function="0x0" slot="0x04" type="pci" />
    </interface>

    <serial type="tcp">
      <source host="127.0.0.1" mode="bind" service="8897" />
      <protocol type="telnet" />
      <target port="0" />
    </serial>

    <console type="tcp">
      <source host="127.0.0.1" mode="bind" service="8897" />
      <protocol type="telnet" />
      <target port="0" type="serial" />
    </console>

    <input bus="usb" type="tablet" />
    <input bus="ps2" type="mouse" />
    <input bus="ps2" type="keyboard" />
    <graphics autoport="yes" listen="127.0.0.1" port="-1" type="vnc">
      <listen address="127.0.0.1" type="address" />
    </graphics>
    <sound model="ac97">
    </sound>
    <video>
      <model heads="1" type="cirrus" vram="9216" />
    </video>

    <memballoon model="virtio">
    </memballoon>

    <interface managed="yes" type="hostdev">        #<------VT-d SR-IOV VF assignment
      <mac address="02:16:0A:0E:FF:31" />           #<------MAC appeared to VM
      <source>                                      #<------VF to be assigned
        <address bus="0x23" domain="0x0000" function="0x0" slot="0x10" type="pci" />
      </source>                                     #<------23:10.0 is p3p1 vf0
    </interface>

    <interface managed="yes" type="hostdev">
      <mac address="02:16:0A:0E:FF:32" />
      <source>
        <address bus="0x06" domain="0x0000" function="0x0" slot="0x10" type="pci" />
      </source>                                     #<------06:10.0 is p2p1 vf0
    </interface>
  </devices>

</domain>
```

generated virtual network XML

br-ext-generated.xml:

```
<network>
  <name>br-ext</name>
  <forward mode="route" />
  <bridge delay="0" name="br-ext" stp="on" />
  <mac address="52:54:00:9f:a0:77" />
  <ip address="10.85.4.17" netmask="255.255.255.128">
    <dhcp>
      <host ip="10.85.4.105" mac="02:04:17:01:01:01" name="vcp-vmx1" />
      <host ip="10.85.4.106" mac="02:04:17:01:01:02" name="vfp-vmx1" />
    </dhcp>
  </ip>
</network>
```

br-int-generated.xml:

```
<network>
  <name>br-int-vmx1</name>
  <bridge delay="0" name="br-int-vmx1" stp="on" />
</network>
```

generated shell scripts

build/vmx1/xml/cpu_affinitize.sh*:*

```
virsh vcpupin vfp-vmx1 0 11
virsh vcpupin vfp-vmx1 1 12
virsh vcpupin vfp-vmx1 2 13
virsh vcpupin vfp-vmx1 3 8
virsh vcpupin vfp-vmx1 4 9
virsh vcpupin vcp-vmx1 0 15
virsh emulatorpin vcp-vmx1 0
virsh emulatorpin vfp-vmx1 0
```

build/vmx1/xml/vfconfig-generated.sh

```
#Handling interface p3p1
ifconfig p3p1 up
sleep 2
ifconfig p3p1 promisc
ifconfig p3p1 allmulti
ifconfig p3p1 mtu 2000
ip link set p3p1 vf 0 mac 02:04:17:01:02:01
ip link set p3p1 vf 0 rate 10000
echo 0000:23:10.0 > /sys/bus/pci/devices/0000:23:10.0/driver/unbind
echo 0000:23:10.0 >> /sys/bus/pci/drivers/pci-stub/bind
ip link set p3p1 vf 0 spoofchk off

#Handling interface p2p1
ifconfig p2p1 up
sleep 2
ifconfig p2p1 promisc
ifconfig p2p1 allmulti
ifconfig p2p1 mtu 2000
ip link set p2p1 vf 0 mac 02:04:17:01:02:02
ip link set p2p1 vf 0 rate 10000
echo 0000:06:10.0 > /sys/bus/pci/devices/0000:06:10.0/driver/unbind
echo 0000:06:10.0 >> /sys/bus/pci/drivers/pci-stub/bind
ip link set p2p1 vf 0 spoofchk off
```

generated files for VIRTIO

comparing with SRIOV, the generated files for VIRTIO remains the same except:

1. the "interface" part for vPFE VM now looks:

```
......
<interface type="network">
  <mac address="02:04:17:01:02:01" />
  <source network="default" />
  <model type="virtio" />
  <target dev="ge-0.0.0-vmx1" />
</interface>
<interface type="network">
  <mac address="02:04:17:01:02:02" />
  <source network="default" />
  <model type="virtio" />
  <target dev="ge-0.0.1-vmx1" />
</interface>
```

2. since there is no concept of VF for VIRTIO virtual interface, there is no script generated for VF configuration.

dumpxml vcp-vmx1

```
ping@trinity:/virtualization/vmx1$ cat vRE-generated-all.xml
<domain type='kvm' id='2'>
  <name>vcp-vmx1</name>
  <uuid>956ce015-cf26-4752-8679-6925f512870c</uuid>
  <memory unit='KiB'>2000896</memory>
  <currentMemory unit='KiB'>2000000</currentMemory>
  <vcpu placement='static'>1</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='15'/>
    <emulatorpin cpuset='0'/>
  </cputune>
  <resource>
    <partition>/machine</partition>
  </resource>
    <sysinfo type='smbios'>
      <bios>
        <entry name='vendor'>Juniper</entry>
      </bios>
      <system>
        <entry name='manufacturer'>VMX</entry>
        <entry name='product'>VM-vcp_vmx1-161-re-0</entry>
        <entry name='version'>0.1.0</entry>
      </system>
    </sysinfo>
  <os>
    <type arch='x86_64' machine='pc-0.13'>hvm</type>
    <boot dev='hd'/>
    <smbios mode='sysinfo'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <cpu mode='host-model'>
    <model fallback='allow'/>
    <topology sockets='1' cores='1' threads='1'/>
  </cpu>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='directsync'/>
      <source file=
'/virtualization/images/vmx_20151102.0/build/vmx1/images/jinstall64-vmx-15.1F-
```

```
20151104.0-domestic.img'/>
      <backingStore/>
      <target dev='hda' bus='ide'/>
      <alias name='ide0-0-0'/>
      <address type='drive' controller='0' bus='0' target='0' unit='0'/>
    </disk>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='directsync'/>
      <source file='
/virtualization/images/vmx_20151102.0/build/vmx1/images/vmxhdd.img'/>
      <backingStore/>
      <target dev='hdb' bus='ide'/>
      <alias name='ide0-0-1'/>
      <address type='drive' controller='0' bus='0' target='0' unit='1'/>
    </disk>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='directsync'/>
      <source file='/virtualization/images/vmx_20151102.0/images/metadata_usb.img'/>
      <backingStore/>
      <target dev='sda' bus='usb'/>
      <alias name='usb-disk0'/>
    </disk>
    <controller type='usb' index='0'>
      <alias name='usb0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x2'/>
    </controller>
    <controller type='ide' index='0'>
      <alias name='ide0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
    </controller>
    <controller type='pci' index='0' model='pci-root'>
      <alias name='pci.0'/>
    </controller>
    <interface type='bridge'>
      <mac address='02:04:17:01:01:01'/>
      <source bridge='br-ext'/>
      <target dev='vcp_ext-vmx1'/>
      <model type='e1000'/>
      <alias name='net0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
    <interface type='bridge'>
      <mac address='52:54:00:7a:91:1a'/>
      <source bridge='br-int-vmx1'/>
      <target dev='vcp_int-vmx1'/>
      <model type='virtio'/>
      <alias name='net1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </interface>
    <serial type='tcp'>
      <source mode='bind' host='127.0.0.1' service='8816'/>
```

```
      <protocol type='telnet'/>
      <target port='0'/>
      <alias name='serial0'/>
    </serial>
    <console type='tcp'>
      <source mode='bind' host='127.0.0.1' service='8816'/>
      <protocol type='telnet'/>
      <target type='serial' port='0'/>
      <alias name='serial0'/>
    </console>
    <input type='tablet' bus='usb'>
      <alias name='input0'/>
    </input>
    <input type='mouse' bus='ps2'/>
    <input type='keyboard' bus='ps2'/>
    <graphics type='vnc' port='5900' autoport='yes' listen='127.0.0.1'>
      <listen type='address' address='127.0.0.1'/>
    </graphics>
    <sound model='ac97'>
      <alias name='sound0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </sound>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
      <alias name='video0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </video>
    <memballoon model='virtio'>
      <alias name='balloon0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

dumpxml vfp-vmx1

```
ping@trinity:/virtualization/vmx1$ cat vPFE-generated-all.xml
<domain type='kvm' id='3'>
  <name>vfp-vmx1</name>
  <uuid>399d395f-d583-42e5-ace9-86118b346565</uuid>
  <memory unit='KiB'>16000000</memory>
  <currentMemory unit='KiB'>16000000</currentMemory>
  <memoryBacking>
    <hugepages/>
    <nosharepages/>
  </memoryBacking>
  <vcpu placement='static'>4</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='11'/>
```

```
    <vcpupin vcpu='1' cpuset='12'/>
    <vcpupin vcpu='2' cpuset='13'/>
    <vcpupin vcpu='3' cpuset='8'/>
    <emulatorpin cpuset='0'/>
  </cputune>
  <numatune>
    <memory mode='strict' nodeset='1'/>
  </numatune>
  <resource>
    <partition>/machine</partition>
  </resource>
  <os>
    <type arch='x86_64' machine='pc-i440fx-trusty'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
  </features>
  <cpu mode='host-model'>
    <model fallback='allow'/>
    <topology sockets='1' cores='4' threads='1'/>
  </cpu>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='directsync'/>
      <source file='/virtualization/images/vmx_20151102.0/build/vmx1/images/vFPC-
20151102.img'/>
      <backingStore/>
      <target dev='hda' bus='ide'/>
      <alias name='ide0-0-0'/>
      <address type='drive' controller='0' bus='0' target='0' unit='0'/>
    </disk>
    <controller type='pci' index='0' model='pci-root'>
      <alias name='pci.0'/>
    </controller>
    <controller type='usb' index='0'>
      <alias name='usb0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x2'/>
    </controller>
    <controller type='ide' index='0'>
      <alias name='ide0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
    </controller>
    <interface type='bridge'>
      <mac address='02:04:17:01:01:02'/>
      <source bridge='br-ext'/>
```

```
    <target dev='vfp_ext-vmx1'/>
    <model type='virtio'/>
    <alias name='net0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  <interface type='bridge'>
    <mac address='52:54:00:9c:b3:f2'/>
    <source bridge='br-int-vmx1'/>
    <target dev='vfp_int-vmx1'/>
    <model type='virtio'/>
    <alias name='net1'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
  </interface>
  <interface type='hostdev' managed='yes'>
    <mac address='02:04:17:01:02:01'/>
    <driver name='kvm'/>
    <source>
      <address type='pci' domain='0x0000' bus='0x23' slot='0x10' function='0x0'/>
    </source>
    <alias name='hostdev0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
  </interface>
  <interface type='hostdev' managed='yes'>
    <mac address='02:04:17:01:02:02'/>
    <driver name='kvm'/>
    <source>
      <address type='pci' domain='0x0000' bus='0x06' slot='0x10' function='0x0'/>
    </source>
    <alias name='hostdev1'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
  </interface>
  <serial type='tcp'>
    <source mode='bind' host='127.0.0.1' service='8817'/>
    <protocol type='telnet'/>
    <target port='0'/>
    <alias name='serial0'/>
  </serial>
  <console type='tcp'>
    <source mode='bind' host='127.0.0.1' service='8817'/>
    <protocol type='telnet'/>
    <target type='serial' port='0'/>
    <alias name='serial0'/>
  </console>
  <input type='tablet' bus='usb'>
    <alias name='input0'/>
  </input>
  <input type='mouse' bus='ps2'/>
  <input type='keyboard' bus='ps2'/>
  <graphics type='vnc' port='5901' autoport='yes' listen='127.0.0.1'>
    <listen type='address' address='127.0.0.1'/>
  </graphics>
```

```
    <sound model='ac97'>
      <alias name='sound0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </sound>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
      <alias name='video0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </video>
    <memballoon model='virtio'>
      <alias name='balloon0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

`virsh capabilities` complete output

```
ping@trinity:~$ sudo virsh capabilities
<capabilities>

  <host>
    <uuid>36373931-3138-5553-4534-333739575353</uuid>
    <cpu>
      <arch>x86_64</arch>
      <model>SandyBridge</model>
      <vendor>Intel</vendor>
      <topology sockets='1' cores='8' threads='1'/>
      <feature name='invtsc'/>
      <feature name='erms'/>
      <feature name='smep'/>
      <feature name='fsgsbase'/>
      <feature name='pdpe1gb'/>
      <feature name='rdrand'/>
      <feature name='f16c'/>
      <feature name='osxsave'/>
      <feature name='dca'/>
      <feature name='pcid'/>
      <feature name='pdcm'/>
      <feature name='xtpr'/>
      <feature name='tm2'/>
      <feature name='est'/>
      <feature name='smx'/>
      <feature name='vmx'/>
      <feature name='ds_cpl'/>
      <feature name='monitor'/>
      <feature name='dtes64'/>
      <feature name='pbe'/>
      <feature name='tm'/>
      <feature name='ht'/>
      <feature name='ss'/>
      <feature name='acpi'/>
      <feature name='ds'/>
      <feature name='vme'/>
      <pages unit='KiB' size='4'/>
      <pages unit='KiB' size='2048'/>
    </cpu>
    <power_management>
      <suspend_disk/>
      <suspend_hybrid/>
    </power_management>
    <migration_features>
      <live/>
      <uri_transports>
        <uri_transport>tcp</uri_transport>
```

```
      </uri_transports>
    </migration_features>
    <topology>
      <cells num='4'>
        <cell id='0'>
          <memory unit='KiB'>132067432</memory>
          <pages unit='KiB' size='4'>33016858</pages>
          <pages unit='KiB' size='2048'>0</pages>
          <distances>
            <sibling id='0' value='10'/>
            <sibling id='1' value='21'/>
            <sibling id='2' value='21'/>
            <sibling id='3' value='21'/>
          </distances>
          <cpus num='8'>
            <cpu id='0' socket_id='0' core_id='0' siblings='0'/>
            <cpu id='1' socket_id='0' core_id='1' siblings='1'/>
            <cpu id='2' socket_id='0' core_id='2' siblings='2'/>
            <cpu id='3' socket_id='0' core_id='3' siblings='3'/>
            <cpu id='4' socket_id='0' core_id='4' siblings='4'/>
            <cpu id='5' socket_id='0' core_id='5' siblings='5'/>
            <cpu id='6' socket_id='0' core_id='6' siblings='6'/>
            <cpu id='7' socket_id='0' core_id='7' siblings='7'/>
          </cpus>
        </cell>
        <cell id='1'>
          <memory unit='KiB'>132116676</memory>
          <pages unit='KiB' size='4'>33029169</pages>
          <pages unit='KiB' size='2048'>0</pages>
          <distances>
            <sibling id='0' value='21'/>
            <sibling id='1' value='10'/>
            <sibling id='2' value='21'/>
            <sibling id='3' value='21'/>
          </distances>
          <cpus num='8'>
            <cpu id='8' socket_id='1' core_id='0' siblings='8'/>
            <cpu id='9' socket_id='1' core_id='1' siblings='9'/>
            <cpu id='10' socket_id='1' core_id='2' siblings='10'/>
            <cpu id='11' socket_id='1' core_id='3' siblings='11'/>
            <cpu id='12' socket_id='1' core_id='4' siblings='12'/>
            <cpu id='13' socket_id='1' core_id='5' siblings='13'/>
            <cpu id='14' socket_id='1' core_id='6' siblings='14'/>
            <cpu id='15' socket_id='1' core_id='7' siblings='15'/>
          </cpus>
        </cell>
        <cell id='2'>
          <memory unit='KiB'>132116676</memory>
          <pages unit='KiB' size='4'>33029169</pages>
          <pages unit='KiB' size='2048'>0</pages>
          <distances>
```

```
          <sibling id='0' value='21'/>
          <sibling id='1' value='21'/>
          <sibling id='2' value='10'/>
          <sibling id='3' value='21'/>
        </distances>
        <cpus num='8'>
          <cpu id='16' socket_id='2' core_id='0' siblings='16'/>
          <cpu id='17' socket_id='2' core_id='1' siblings='17'/>
          <cpu id='18' socket_id='2' core_id='2' siblings='18'/>
          <cpu id='19' socket_id='2' core_id='3' siblings='19'/>
          <cpu id='20' socket_id='2' core_id='4' siblings='20'/>
          <cpu id='21' socket_id='2' core_id='5' siblings='21'/>
          <cpu id='22' socket_id='2' core_id='6' siblings='22'/>
          <cpu id='23' socket_id='2' core_id='7' siblings='23'/>
        </cpus>
      </cell>
      <cell id='3'>
        <memory unit='KiB'>132116616</memory>
        <pages unit='KiB' size='4'>33029154</pages>
        <pages unit='KiB' size='2048'>0</pages>
        <distances>
          <sibling id='0' value='21'/>
          <sibling id='1' value='21'/>
          <sibling id='2' value='21'/>
          <sibling id='3' value='10'/>
        </distances>
        <cpus num='8'>
          <cpu id='24' socket_id='3' core_id='0' siblings='24'/>
          <cpu id='25' socket_id='3' core_id='1' siblings='25'/>
          <cpu id='26' socket_id='3' core_id='2' siblings='26'/>
          <cpu id='27' socket_id='3' core_id='3' siblings='27'/>
          <cpu id='28' socket_id='3' core_id='4' siblings='28'/>
          <cpu id='29' socket_id='3' core_id='5' siblings='29'/>
          <cpu id='30' socket_id='3' core_id='6' siblings='30'/>
          <cpu id='31' socket_id='3' core_id='7' siblings='31'/>
        </cpus>
      </cell>
    </cells>
  </topology>
  <secmodel>
    <model>none</model>
    <doi>0</doi>
  </secmodel>
  <secmodel>
    <model>dac</model>
    <doi>0</doi>
    <baselabel type='kvm'>+0:+0</baselabel>
    <baselabel type='qemu'>+0:+0</baselabel>
  </secmodel>
</host>
```

```
  <guest>
    <os_type>hvm</os_type>
    <arch name='i686'>
      <wordsize>32</wordsize>
      <emulator>/usr/bin/qemu-system-i386</emulator>
      <machine canonical='pc-i440fx-trusty' maxCpus='255'>pc</machine>
      <machine maxCpus='255'>pc-0.12</machine>
      <machine maxCpus='255'>pc-1.3</machine>
      <machine maxCpus='255'>pc-q35-1.6</machine>
      <machine canonical='pc-1.0-qemu-kvm' maxCpus='255'>pc-1.0-precise</machine>
      <machine maxCpus='255'>pc-q35-1.5</machine>
      <machine maxCpus='1'>xenpv</machine>
      <machine maxCpus='255'>pc-i440fx-1.6</machine>
      <machine maxCpus='255'>pc-i440fx-1.7</machine>
      <machine canonical='pc-i440fx-1.5-qemu-kvm' maxCpus='255'>pc-i440fx-1.5-
saucy</machine>
      <machine maxCpus='255'>pc-0.11</machine>
      <machine maxCpus='255'>pc-0.10</machine>
      <machine maxCpus='255'>pc-1.2</machine>
      <machine maxCpus='1'>isapc</machine>
      <machine maxCpus='255'>pc-q35-1.4</machine>
      <machine maxCpus='128'>xenfv</machine>
      <machine maxCpus='255'>pc-0.15</machine>
      <machine maxCpus='255'>pc-0.14</machine>
      <machine maxCpus='255'>pc-i440fx-1.5</machine>
      <machine canonical='pc-q35-2.0' maxCpus='255'>q35</machine>
      <machine maxCpus='255'>pc-i440fx-1.4</machine>
      <machine maxCpus='255'>pc-1.1</machine>
      <machine maxCpus='255'>pc-q35-1.7</machine>
      <machine canonical='pc-1.0' maxCpus='255'>pc-1.0-qemu-kvm</machine>
      <machine maxCpus='255'>pc-i440fx-2.0</machine>
      <machine maxCpus='255'>pc-0.13</machine>
      <domain type='qemu'>
      </domain>
      <domain type='kvm'>
        <emulator>/usr/bin/kvm</emulator>
        <machine canonical='pc-i440fx-trusty' maxCpus='255'>pc</machine>
        <machine maxCpus='255'>pc-1.3</machine>
        <machine maxCpus='255'>pc-0.12</machine>
        <machine maxCpus='255'>pc-q35-1.6</machine>
        <machine canonical='pc-1.0-qemu-kvm' maxCpus='255'>pc-1.0-precise</machine>
        <machine maxCpus='255'>pc-q35-1.5</machine>
        <machine maxCpus='1'>xenpv</machine>
        <machine maxCpus='255'>pc-i440fx-1.6</machine>
        <machine canonical='pc-i440fx-1.5-qemu-kvm' maxCpus='255'>pc-i440fx-1.5-
saucy</machine>
        <machine maxCpus='255'>pc-i440fx-1.7</machine>
        <machine maxCpus='255'>pc-0.11</machine>
        <machine maxCpus='255'>pc-1.2</machine>
        <machine maxCpus='255'>pc-0.10</machine>
        <machine maxCpus='1'>isapc</machine>
```

```
        <machine maxCpus='255'>pc-q35-1.4</machine>
        <machine maxCpus='128'>xenfv</machine>
        <machine maxCpus='255'>pc-0.15</machine>
        <machine maxCpus='255'>pc-0.14</machine>
        <machine maxCpus='255'>pc-i440fx-1.5</machine>
        <machine maxCpus='255'>pc-i440fx-1.4</machine>
        <machine canonical='pc-q35-2.0' maxCpus='255'>q35</machine>
        <machine maxCpus='255'>pc-1.1</machine>
        <machine maxCpus='255'>pc-q35-1.7</machine>
        <machine canonical='pc-1.0' maxCpus='255'>pc-1.0-qemu-kvm</machine>
        <machine maxCpus='255'>pc-i440fx-2.0</machine>
        <machine maxCpus='255'>pc-0.13</machine>
      </domain>
    </arch>
    <features>
      <cpuselection/>
      <deviceboot/>
      <disksnapshot default='on' toggle='no'/>
      <acpi default='on' toggle='yes'/>
      <apic default='on' toggle='no'/>
      <pae/>
      <nonpae/>
    </features>
  </guest>

  <guest>
    <os_type>hvm</os_type>
    <arch name='x86_64'>
      <wordsize>64</wordsize>
      <emulator>/usr/bin/qemu-system-x86_64</emulator>
      <machine canonical='pc-i440fx-trusty' maxCpus='255'>pc</machine>
      <machine maxCpus='255'>pc-1.3</machine>
      <machine maxCpus='255'>pc-0.12</machine>
      <machine maxCpus='255'>pc-q35-1.6</machine>
      <machine canonical='pc-1.0-qemu-kvm' maxCpus='255'>pc-1.0-precise</machine>
      <machine maxCpus='255'>pc-q35-1.5</machine>
      <machine maxCpus='1'>xenpv</machine>
      <machine maxCpus='255'>pc-i440fx-1.6</machine>
      <machine canonical='pc-i440fx-1.5-qemu-kvm' maxCpus='255'>pc-i440fx-1.5-
saucy</machine>
      <machine maxCpus='255'>pc-i440fx-1.7</machine>
      <machine maxCpus='255'>pc-0.11</machine>
      <machine maxCpus='255'>pc-1.2</machine>
      <machine maxCpus='255'>pc-0.10</machine>
      <machine maxCpus='1'>isapc</machine>
      <machine maxCpus='255'>pc-q35-1.4</machine>
      <machine maxCpus='128'>xenfv</machine>
      <machine maxCpus='255'>pc-0.15</machine>
      <machine maxCpus='255'>pc-0.14</machine>
      <machine maxCpus='255'>pc-i440fx-1.5</machine>
      <machine maxCpus='255'>pc-i440fx-1.4</machine>
```

```
      <machine canonical='pc-q35-2.0' maxCpus='255'>q35</machine>
      <machine maxCpus='255'>pc-1.1</machine>
      <machine maxCpus='255'>pc-q35-1.7</machine>
      <machine canonical='pc-1.0' maxCpus='255'>pc-1.0-qemu-kvm</machine>
      <machine maxCpus='255'>pc-i440fx-2.0</machine>
      <machine maxCpus='255'>pc-0.13</machine>
      <domain type='qemu'>
      </domain>
      <domain type='kvm'>
        <emulator>/usr/bin/kvm</emulator>
        <machine canonical='pc-i440fx-trusty' maxCpus='255'>pc</machine>
        <machine maxCpus='255'>pc-1.3</machine>
        <machine maxCpus='255'>pc-0.12</machine>
        <machine maxCpus='255'>pc-q35-1.6</machine>
        <machine canonical='pc-1.0-qemu-kvm' maxCpus='255'>pc-1.0-precise</machine>
        <machine maxCpus='255'>pc-q35-1.5</machine>
        <machine maxCpus='1'>xenpv</machine>
        <machine maxCpus='255'>pc-i440fx-1.6</machine>
        <machine canonical='pc-i440fx-1.5-qemu-kvm' maxCpus='255'>pc-i440fx-1.5-
saucy</machine>
        <machine maxCpus='255'>pc-i440fx-1.7</machine>
        <machine maxCpus='255'>pc-0.11</machine>
        <machine maxCpus='255'>pc-1.2</machine>
        <machine maxCpus='255'>pc-0.10</machine>
        <machine maxCpus='1'>isapc</machine>
        <machine maxCpus='255'>pc-q35-1.4</machine>
        <machine maxCpus='128'>xenfv</machine>
        <machine maxCpus='255'>pc-0.15</machine>
        <machine maxCpus='255'>pc-0.14</machine>
        <machine maxCpus='255'>pc-i440fx-1.5</machine>
        <machine maxCpus='255'>pc-i440fx-1.4</machine>
        <machine canonical='pc-q35-2.0' maxCpus='255'>q35</machine>
        <machine maxCpus='255'>pc-1.1</machine>
        <machine maxCpus='255'>pc-q35-1.7</machine>
        <machine canonical='pc-1.0' maxCpus='255'>pc-1.0-qemu-kvm</machine>
        <machine maxCpus='255'>pc-i440fx-2.0</machine>
        <machine maxCpus='255'>pc-0.13</machine>
      </domain>
    </arch>
    <features>
      <cpuselection/>
      <deviceboot/>
      <disksnapshot default='on' toggle='no'/>
      <acpi default='on' toggle='yes'/>
      <apic default='on' toggle='no'/>
    </features>
  </guest>

</capabilities>
```

ixgbe driver issue

The orignal IXGBE driver coming with ubuntu has issue on multicast - the VMX "Getting Started Guide" does mention this:

> Multicast promiscuous mode for Virtual Functions is needed to receive control traffic that comes with broadcast MAC addresses

So it looks like an issue of multicast support specifically on VF. A quick test here illustates the issue.

To test this I manually setup a VMX, with the original IXGBE driver coming with ubuntu. the setup is as simple as just a HP server with VMX installed connected to another L3 switch(QFX), showing below:

ixgbe multicast issue test diagram

```
   HP server
.............
.  +------+ .                   +------+
.  |      | .                   |      |
.  |VMX   | .TX RX              |QFX   |
.  |      | . |  |              |      |
.  +------+ . |  |              |      |
.           . |  |              |      |
.10.10.10.1 . |  X              |      |
.  NIC/VF   . |  |              |      |
.  +------+ . v  +-<-----------  |      |
.  +---+--+ . |                 |      |
.      |    . +---->-----------  |      |
.............                    +--+---+
       |                           | 10.10.10.2
       +---------------------------+
```

configuration on both end are simple: an IP interface with OSPF enabled on it.

QFX configuration

```
set groups test-ixgbe interfaces xe-0/0/18 unit 0 family inet address 10.10.10.2/24
set groups test-ixgbe routing-instances test instance-type virtual-router
set groups test-ixgbe routing-instances test interface xe-0/0/18.0
set groups test-ixgbe routing-instances test protocols ospf area 0.0.0.0 interface xe-
0/0/18.0
```

VMX configuration

```
root# show | compare
[edit]
+  protocols {
+      ospf {
+          area 0.0.0.0 {
+              interface ge-0/0/0.0;
+          }
+      }
+  }
```

```
[edit]
root# run show ospf interface
Interface           State   Area            DR ID           BDR ID          Nbrs
ge-0/0/0.0          DR      0.0.0.0         10.10.10.1      0.0.0.0            0
```

the issue

the issue is that ping works, but OSPF adjacency does not come up:

```
vmx# run ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2): 56 data bytes
64 bytes from 10.10.10.2: icmp_seq=0 ttl=64 time=13.889 ms
^C
--- 10.10.10.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 13.889/13.889/13.889/0.000 ms
```

```
root# root# run show ospf neighbor
root#
```

packet capture

packet capture on VMX host server shows physical port (p2p1) is able to:

- receive ospf packets from peer device
- deliver OSPF packets out once receiving from VMX

```
ping@matrix:/home/vAVPN/images$ sudo tcpdump -ni p2p1
tcpdump: WARNING: p2p1: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on p2p1, link-type EN10MB (Ethernet), capture size 65535 bytes
12:18:12.963197 IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
12:18:17.477019 IP 10.10.10.2 > 224.0.0.5: OSPFv2, Hello, length 60
12:18:21.483668 IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
12:18:21.989157 LLDP, length 296: sonata
^C
```

but packet capture from VMX shows it doesn't receive anything:

```
root# run monitor traffic interface ge-0/0/0 size 2000
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is ON. Use <no-resolve> to avoid any reverse lookup delay.
Address resolution timeout is 4s.
Listening on ge-0/0/0, capture size 2000 bytes
```

```
Reverse lookup for 224.0.0.5 failed (check DNS reachability).
Other reverse lookup failures will not be reported.
Use <no-resolve> to avoid reverse lookups on IP addresses.
```

```
20:17:20.318417 Out IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
20:17:29.588411 Out IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
20:17:37.958421 Out IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
20:17:47.058547 Out IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
20:17:54.598639 Out IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
20:18:03.078845 Out IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
20:18:12.408862 Out IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
20:18:20.929166 Out IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
```

meanwhile capture in QFX shows it is receiving and sending packets without a problem:

```
labroot@sonata# run monitor traffic interface xe-0/0/18
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is ON. Use <no-resolve> to avoid any reverse lookup delay.
Address resolution timeout is 4s.
Listening on xe-0/0/18, capture size 96 bytes
```

```
Reverse lookup for 224.0.0.5 failed (check DNS reachability).
Other reverse lookup failures will not be reported.
Use <no-resolve> to avoid reverse lookups on IP addresses.
```

```
16:00:25.375850  In IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
16:00:28.511860 Out IP truncated-ip - 20 bytes missing! 10.10.10.2 > 224.0.0.5:
OSPFv2, Hello, length 60
16:00:30.867355 Out LLDP, name sonata, length 60
        [|LLDP]
16:00:35.119801  In IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
16:00:37.549066 Out IP truncated-ip - 20 bytes missing! 10.10.10.2 > 224.0.0.5:
OSPFv2, Hello, length 60
16:00:42.633445  In IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
16:00:47.589120 Out IP truncated-ip - 20 bytes missing! 10.10.10.2 > 224.0.0.5:
OSPFv2, Hello, length 60
16:00:52.077874  In IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
16:00:56.129699 Out IP truncated-ip - 20 bytes missing! 10.10.10.2 > 224.0.0.5:
OSPFv2, Hello, length 60
16:00:58.003400 Out LLDP, name sonata, length 60
        [|LLDP]
16:01:01.841421  In IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
16:01:05.589241 Out IP truncated-ip - 20 bytes missing! 10.10.10.2 > 224.0.0.5:
OSPFv2, Hello, length 60
16:01:11.136212  In IP 10.10.10.1 > 224.0.0.5: OSPFv2, Hello, length 56
16:01:14.940515 Out IP truncated-ip - 20 bytes missing! 10.10.10.2 > 224.0.0.5:
OSPFv2, Hello, length 60
```

So it just looks like OSPF packet coming in the server NIC does not get handed over to VMX. `ethtool` with `-S` option will print counters for all VFs, here is the capture when the issue is ongoing:

```
ping@matrix:~$ ethtool -S p2p1  | grep -iE "rx_packets|tx_packets|VF 0"
     rx_packets: 33685
     tx_packets: 8
     VF 0 Rx Packets: 315
     VF 0 Rx Bytes: 19546
     VF 0 Tx Packets: 20407
     VF 0 Tx Bytes: 1821654
     VF 0 MC Packets: 0
```

It shows VF 0 didn't receive any packet. Given that physical NIC p2p1 (PF) actually received the packet, we can conclude all multicast packets were dropped at VF 0.

For physical NIC the multicast capability can be enabled by this command:

```
ifconfig p3p1 up promisc allmulti mtu 2000
```

unfortunately this does not apply to VFs. The current solution is Juniper modify the IXGBE code to fix this. So recompiling IXGBE driver from Juniper provided source code will solve this issue.

end