Introduction to VPC-SI

This chapter introduces Cisco Virtualized Packet Core—Single Instance (VPC-SI). VPC-SI addresses the need for virtualized cloud architectures that enable the accelerated deployment of new applications and services in the mobile market.

Product Description

This chapter describes the StarOS VPC-SI architecture and interaction with external devices.

VPC-SI consolidates the operations of physical Cisco ASR 5500 chassis running StarOS into a single Virtual Machine (VM) able to run on commercial off-the-shelf (COTS) servers. Each VPC-SI VM operates as an independent StarOS instance, incorporating the management and session processing capabilities of a physical chassis.

Virtualized Mobility Functions

VPC-SI consists of the set virtualized mobility functions that implement mobility specific services and applications within the core of the network. These functions include the:

- Mobile Packet Core
  - LTE MME (Mobile Management Entity), P-GW (PDN Gateway) and S-GW (Serving Gateway)
  - GGSN Gateway GPRS Support Node
  - SAE-GW System Architecture Evolution Gateway
  - SGSN Serving GPRS Support Node (3G only)

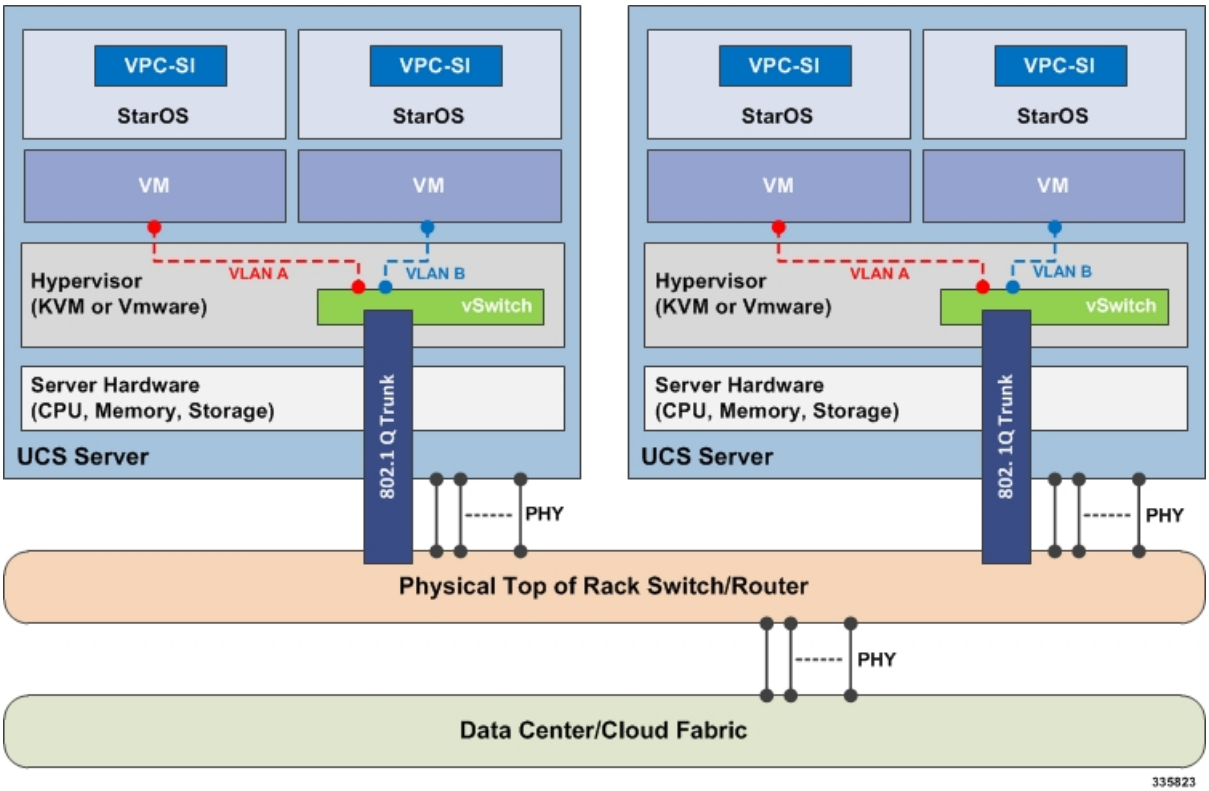
- Mobile Control Plane PCRF (Policy and Charging Rule Function), application gateway, analytics, services orchestration, abstraction and control functions
- Small cell gateways
  - HNBGW Home NodeB Gateway
  - HeNBGW evolved Home NodeB Gateway
  - SAMOG S2a Mobility over GTP combine CGW (Converged Access Gateway) and Trusted WLAN AAA Proxy (TWAP) functions on a single service node

Mobile Cloud Network (MCN) is a network infrastructure that includes Infrastructure as a Service (IaaS), the orchestration mechanisms, analytics mechanisms etc., upon which the VPC-SI as well as other services are deployed.

VM Interconnect Architecture

This figure below shows basic L2/L3 interconnection as supported by VPC-SI.

Figure 1: L2/L3 Interconnection



In the figure above, a virtual switch is embedded within the hypervisor to support SDN L2 capabilities across the data center. The virtual switch is interconnected to other virtual switches using 802.1Q trunks (VLANs). Typically, the virtual switch is a dynamically loaded kernel module.

Standalone Instance

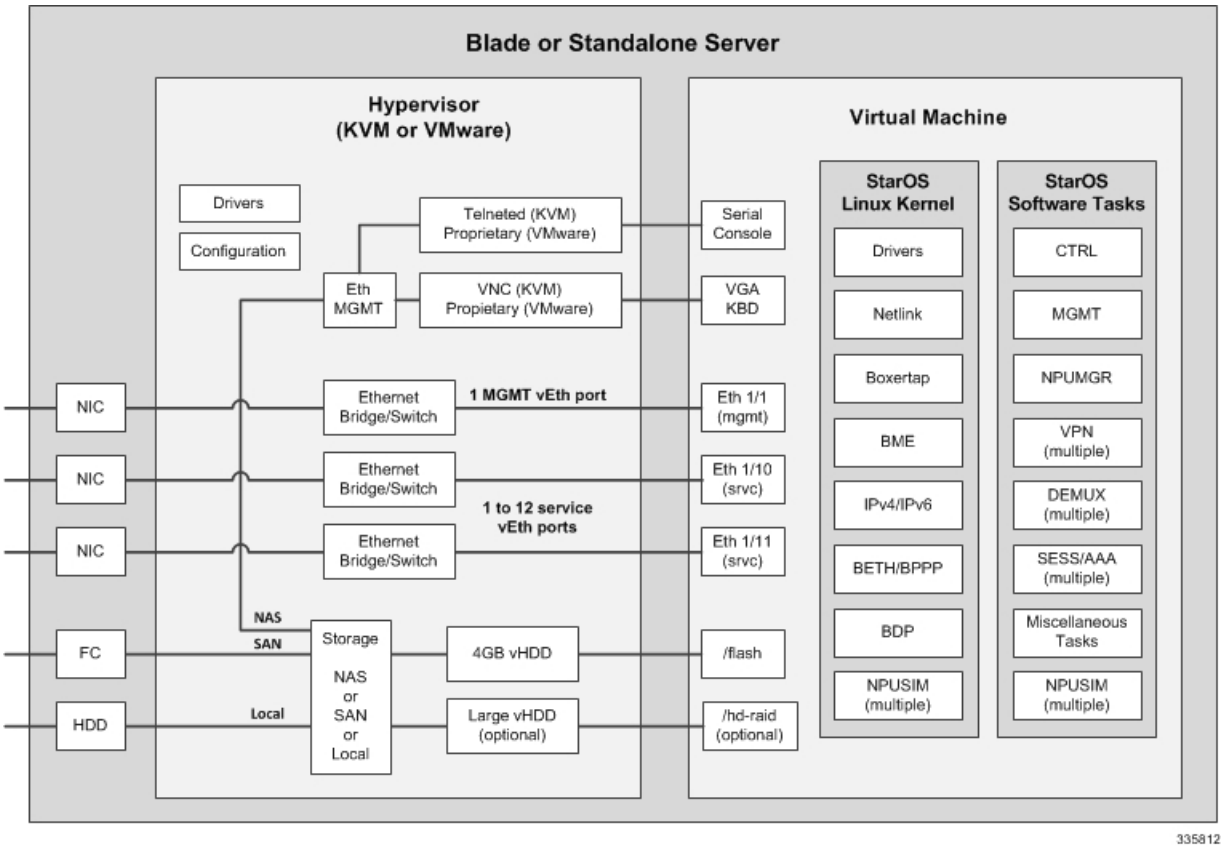
VPC-SI is essentially StarOS running within a Virtual Machine (VM) on a COTS platform. It can be used as a stand-alone single VM within an enterprise, remote site, or customer data center. Alternatively, VPC-SI can be integrated as a part of a larger service provider orchestration solution.

The Single Instance architecture is best suited for low capacity scenarios. Scaling the VPC-SI Virtual Network Function (VNF) requires significant network level configurations for certain VNF types (such as, P-GW, S-GW, MME, PCRF, Firewall and NAT). For example, if a new VPC-SI P-GW is added or removed, various Diameter peers must be configured with this information DNS is provisioned or de-provisioned with this information.

VPC-SI only interacts with supported hypervisors KVM (Kernel-based Virtual Machine) and VMware ESXi. It has little or no knowledge of physical devices.

Typically, VPC-SI should be deployed in Interchassis Session Recovery (ICSR) pairs to provide physical redundancy in case of hardware or hypervisor failure.

Figure 2: VPC-SI Overview



Each VPC-SI VM takes on the roles of an entire StarOS system. The only interfaces exposed outside the VM are those for external management and service traffic. Each VM is managed independently.

Each VPC-SI VM performs the following StarOS functions:

- Controller tasks
- Out-of-band management for CLI and Logging (vSerial and vKVM)

- Local context vpnmgr
- Local context management (vNICs)
- System boot image and configuration storage on vHDD
- Record storage on vHDD
- NPU simulation via fastpath and slowpath
- Non-local context (vNICs, 1 to 12).
- Demux and vpnmgr for session processing
- Crypto processing

Feature Set

Interfaces and Addressing

The VM is represented as a virtual card with a single CPU subsystem. This makes many CLI commands, logs, and functions work similarly to StarOS running on ASR 5500 platform.

Applications written for StarOS see VPC-SI as just another platform with a one-slot virtual chassis supporting a single virtual card.

StarOS concepts of contexts, services, pools, interfaces, cards, and ports exist on VPC-SI just as on existing platforms.

When the VM boots, the vNICs configured in the VM profile are detected and an equivalent number of 'Virtual Ethernet' type ports appear in the StarOS CLI.

- VPC-SI assigns the vNIC interfaces in the order offered by the hypervisor.
  - First interface offered is 1/1 for VPC-SI management.
  - Second interface offered is 1/10 for VPC-SI Services control and data traffic.
  - Optional third interface offered is 1/11 for VPC-SI Services control and data traffic.
  - Optional fourth and subsequent interfaces will be 1/12, 1/13
- It is critical to confirm that the interfaces listed in the supported hypervisors line up with the KVM BR group or VMware vSwitch in the order in which you want them to match the VPC-SI interfaces.

Note You cannot be guaranteed that the order of the vNICs as listed in the hypervisor CLI/GUI is the same as how the hypervisor offers them to VPC-SI. On initial setup you must use the **show hardware** CLI command to walk through the MAC addresses shown on the hypervisor's vNIC configuration and match them up with the MAC addresses learned by VPC-SI. This will confirm that the VPC-SI interfaces are connected to the intended BR group/Vswitch.

Encryption

VMs within a VPC-SI instance perform software-based encryption and tunneling of packets (as opposed to the higher-throughput hardware-based services). Call models that make heavy use of encryption for bearer packets or have significant PKI (Public Key Infrastructure) key generation rates may require significant compute resources.

Security

Security of external traffic including tunneling, encryption, Access Control Lists (ACLs), context separation, and user authentication function as on existing StarOS platforms. User ports and interfaces on the CFs and SFs are protected through StarOS CLI configuration.

The virtual system adds additional security concerns on the customer because network communication travel over the DI network on datacenter equipment.

The DI network must be isolated from other hosts within the datacenter by limiting membership in the system network's VLAN to VMs within that specific VPC-SI instance. Unauthorized access to the DI network through other hosts being inadvertently added to that network or the compromise of a router, switch or hypervisor could disrupt or circumvent the security measures of StarOS. Such disruptions can result in failures, loss of service, and/or exposure of control and bearer packets. Properly securing access to the DI network is beyond the control of StarOS.

Communication between DI network component (e.g. CF and SF) VMs is now only possibley via authentication over externally supplied SSH keys. In addition, the system enforces public/private key-based SSH authentication for logins within the DI network. No passwords, keys or LI information are stored or sent in clear text.

If an operator requires physical separation of networks, such as management versus bearer versus LI (Lawful Intercept), then physical separation of the DI network should also be done since it carries sensitive data. In a virtualized environment, the physical separation of networks may not be possible or practical. Operators that have these requirements may need to qualify their hypervisor and infrastructure to confirm that it will provide sufficient protection for their needs.

Redundancy and Availability

Platform Requirements

The virtual system relies on the underlying hardware and hypervisor for overall system redundancy and availability.

The hardware and hypervisor should provide:

- Redundant hardware components where practical (such as power supplies, disks)
- Redundant network paths (dual fabric/NICs, with automatic failover)
- Redundant network uplinks (switches, routers, etc.)

High availability can only be achieved if the underlying infrastructure (hosts, hypervisor, and network) can provide availability and reliability that exceeds expected values. The system is only as reliable as the environment on which it runs.

Interchassis Session Recovery (ICSR) is also recommended to improve availability and recovery time in the case of a non-redundant hardware failure (such as CPU, memory, motherboard, hypervisor software). ICSR provides redundancy at the session level for gateways only.

ICSR Support

VPC-SI supports ICSR between two instances for services that support ICSR in the StarOS software release. When more than one service type is in use, only those services that support ICSR will be able to use ICSR.

ICSR supports redundancy for site/row/rack/host outages, and major software faults. To do so, the two instances should be run on non-overlapping hosts and network interconnects. ICSR is supported only between like-configured instances. ICSR between a VPC-SI instance and another type of platform (such as an ASR 5500) is not supported.

L3 ICSR is supported.

For additional information, refer to the *Interchassis Session Recovery* chapter in this guide.

Hypervisor Requirements

VPC-SI has been qualified to run under the following hypervisors:

- Kernel-based Virtual Machine (KVM) - QEMU emulator 2.0. The VPC-SI StarOS installation build includes a libvirt XML template and ssi_install.sh for VM creation under Ubuntu Server14.04.
- KVM - Red Hat Enterprise Linux 7.2: The VPC-SI StarOS installation build includes an install script called qvpc-si_install.sh.
- VMware ESXi 6.0: The VPC-SI StarOS installation build includes OVF (Open Virtualization Format) and OVA (Open Virtual Application) templates for VM creation via the ESXi GUI.
- VMware ESXi
  - Version 6.0: The VPC-SI StarOS installation build includes OVF (Open Virtualization Format) and OVA (Open Virtual Application) templates for VM creation via the ESXi GUI. This version is supported in releases prior to Release 21.8
  - Version 6.5: Supported in Release 21.8 and 21.9
  - Version 6.7: Supported from Release 21.10 onwards

VM Configuration

VPC-SI requires that the VM be configured with:

- X vCPUs (see vCPU and vRAM Options, on page 7)
- Y vRAM (see vCPU and vRAM Options, on page 7)
- First vNIC is the management port (see vNIC Options, on page 7)
- Second and subsequent vNICs are service ports; one vNIC is required and up to 12 are supported by the VPC, but this number may be limited by the hypervisor
- First vHDD is for boot image and configuration storage (4 GB recommended)
- Second vHDD is for record storage [optional] (16 GB minimum)

vCPU and vRAM Options

A CPU is a single physical computer chip that can have more than one physical CPU core that is fully capable of running the entire system and applications by itself. Virtual core technology supports multiple logical processors (vCPUs) per physical core. The total number of vCPUs supported on a specific platform varies based on the number of available physical cores and the type of virtual core technology implemented in each core.

CF and SF run within VMs that are assigned a number of vCPUs, each supporting one thread (sequence of instructions). The number of available vCPUs supported by the platform CPU may exceed the maximum number of vCPUs that can be assigned to the VM via the hypervisor.

Note The number vCPUs per VM should never exceed the maximum number of vCPUs supported by the platform CPU.

To maximize performance, it may be desirable to adjust the number of vCPUs or vRAM to align with the underlying hardware. SF supports varied vCPU and vRAM combinations, however all SFs must share the same combination within an instance.

Software will determine the optimal number of SESSMGR tasks per SF on startup of the SF based on the number of vCPUs and amount of vRAM on that SF.

Note Dynamic resizing of vCPU count, vRAM size or vNIC type/count (via hotplug, ballooning, etc.) is not supported. If these values need to be changed after provisioning, all VMs must be shut down and reconfigured. Reconfiguration can be performed only on all VMs at once. VMs cannot be reconfigured one at a time since the CPUs and RAM would not match the other instances.

vNIC Options

In this release the supported vNIC options include:

- VMXNET3—Paravirtual NIC for VMware
- VIRTIO—Paravirtual NIC for KMV
- ixgbe—Intel 10 Gigabit NIC virtual function
- enic—Cisco UCS NIC

Hard Drive Storage

In additional to the mandatory /flash (non-RAID) drive, the system supports RAID1 under a virtual machine (VM). For each VM, Virtual SCSI disks can be created, on CF only, matching the SCSI ID shown in this table. The minimum disk size must be greater than 16 GB.

Table 1: Disk Mapping

Type	/flash (non-RAID)	hd-local1	Notes
KVM	SCSI 0:0:0:0	SCSI 0:0:1:0	Raw disk hd-local1 uses RAID1
VMware	SCSI 0:0:0:0	SCSI 0:0:1:0	Raw disk hd-local1 and hd-remote1 use RAID1

For record storage (CDRs and UDRs) the CF VM should be provisioned with a second vHDD sized to meet anticipated record requirements (minimum 16GB). Records will be written to /records on the second vHDD.

DPDK Internal Forwarder

The Intel Data Plane Development Kit (DPDK) is an integral part of the VPC-SI architecture and is used to enhance performance of VPC-SI systems configured with 8 or more vCPUs. The DPDK Internal Forwarder (IFTASK) is a software component that is responsible for packet input and output operations and provides a fast path for packet processing in the user space by bypassing the Linux kernel. During the VPC-SI boot process, a proportion of the vCPUs are allocated to IFTASK and the remainder are allocated to application processing.

To determine which vCPUs are used by IFTASK and view their utilization, use the **show npu utilization table** command as shown here:

```
[local]mySystem# show npu utilization table

******** show npu utilization table card 1 *******
             5-Sec Avg:   lcore00|lcore01|lcore02|lcore03|lcore04|lcore05|lcore06|lcore07|

                     IDLE:           |     4%|    43%|    24%|    45%|    17%|       |
  |
           QUEUE_PORT_RX:           |    34%|    56%|       |       |       |       |
  |
           QUEUE_PORT_TX:           |       |       |       |       |       |       |
  |
           QUEUE_VNPU_RX:           |       |       |       |       |       |       |
  |
           QUEUE_VNPU_TX:           |       |       |       |       |       |       |
  |
            QUEUE_KNI_RX:           |    57%|       |       |       |       |       |
  |
            QUEUE_KNI_TX:           |       |       |       |       |       |       |
  |
       QUEUE_THREAD_KNI:           |     3%|       |       |       |       |       |
  |
          QUEUE_MCDMA_RX:           |       |       |     7%|       |     5%|       |
  |
          QUEUE_MCDMA_TX:           |       |       |       |       |       |       |
  |
     QUEUE_THREAD_MCDMA:           |       |       |     8%|       |    24%|       |
  |
      QUEUE_THREAD_VNPU:           |       |       |       |       |       |       |
  |
         QUEUE_CRYPTO_RX:           |       |       |       |       |       |       |
  |
```

```
       QUEUE_CRYPTO_TX:          |       |       |       |       |       |       |
|
      QUEUE_THREAD_IPC:          |       |       |       |       |       |       |
|
           MCDMA_FLUSH:          |       |       |    59%|    54%|    51%|       |
|
  QUEUE_THREAD_TYPE_MAX:         |       |       |       |       |       |       |
|
         300-Sec Avg:   lcore00|lcore01|lcore02|lcore03|lcore04|lcore05|lcore06|lcore07|

                  IDLE:          |    99%|   100%|    31%|    30%|    32%|       |
|
         QUEUE_PORT_RX:          |     0%|       |       |       |       |       |
|
         QUEUE_PORT_TX:          |       |       |       |       |       |       |
|
         QUEUE_VNPU_RX:          |       |       |       |       |       |       |
|
         QUEUE_VNPU_TX:          |       |       |       |       |       |       |
|
          QUEUE_KNI_RX:          |       |       |       |       |       |       |
|
          QUEUE_KNI_TX:          |       |       |       |       |       |       |
|
      QUEUE_THREAD_KNI:          |       |       |       |       |       |       |
|
        QUEUE_MCDMA_RX:          |       |       |     0%|     0%|     0%|       |
|
        QUEUE_MCDMA_TX:          |       |       |       |       |       |       |
|
    QUEUE_THREAD_MCDMA:          |       |       |       |       |       |       |
|
     QUEUE_THREAD_VNPU:          |       |       |       |       |       |       |
|
       QUEUE_CRYPTO_RX:          |       |       |       |       |       |       |
|
       QUEUE_CRYPTO_TX:          |       |       |       |       |       |       |
|
      QUEUE_THREAD_IPC:          |       |       |       |       |       |       |
|
           MCDMA_FLUSH:          |       |       |    68%|    69%|    67%|       |
|
  QUEUE_THREAD_TYPE_MAX:         |       |       |       |       |       |       |
|
         900-Sec Avg:   lcore00|lcore01|lcore02|lcore03|lcore04|lcore05|lcore06|lcore07|

                  IDLE:          |    99%|   100%|    31%|    31%|    32%|       |
|
         QUEUE_PORT_RX:          |     0%|       |       |       |       |       |
|
         QUEUE_PORT_TX:          |       |       |       |       |       |       |
|
         QUEUE_VNPU_RX:          |       |       |       |       |       |       |
|
         QUEUE_VNPU_TX:          |       |       |       |       |       |       |
|
          QUEUE_KNI_RX:          |       |       |       |       |       |       |
|
          QUEUE_KNI_TX:          |       |       |       |       |       |       |
|
      QUEUE_THREAD_KNI:          |       |       |       |       |       |       |
|
        QUEUE_MCDMA_RX:          |       |       |     0%|     0%|     0%|       |
|
```

```
          QUEUE_MCDMA_TX:           |       |       |       |       |       |       |
|
      QUEUE_THREAD_MCDMA:           |       |       |       |       |       |       |
|
       QUEUE_THREAD_VNPU:           |       |       |       |       |       |       |
|
         QUEUE_CRYPTO_RX:           |       |       |       |       |       |       |
|
         QUEUE_CRYPTO_TX:           |       |       |       |       |       |       |
|
        QUEUE_THREAD_IPC:           |       |       |       |       |       |       |
|
             MCDMA_FLUSH:           |       |       |   68%|    68%|    67%|       |
|
   QUEUE_THREAD_TYPE_MAX:           |       |       |       |       |       |       |
|
```

```
thread 2 QUEUE_PORT_RX                  77.22 %
thread 2 IDLE                           22.78 %
-----------------------------------------------
thread 5 MCDMA_FLUSH                    57.74 %
thread 5 IDLE                           40.13 %
thread 5 QUEUE_THREAD_MCDMA              2.13 %
-----------------------------------------------
thread 1 QUEUE_KNI_RX                   50.39 %
thread 1 QUEUE_PORT_RX                  40.72 %
thread 1 IDLE                            6.13 %
thread 1 QUEUE_THREAD_KNI                2.76 %
-----------------------------------------------
thread 3 QUEUE_THREAD_MCDMA             41.17 %
thread 3 MCDMA_FLUSH                    38.31 %
thread 3 IDLE                           16.28 %
thread 3 QUEUE_MCDMA_RX                  4.24 %
-----------------------------------------------
thread 4 IDLE                           56.03 %
thread 4 MCDMA_FLUSH                    43.97 %
-----------------------------------------------
```

To view CPU utilization for the VM without the IFTASK vCPUs, use the **show cpu info** command. For more detailed information use the **verbose** keyword.

```
[local]mySystem# show cpu info
Card 1, CPU 0:
  Status                : Active, Kernel Running, Tasks Running
  Load Average          : 8.99, 9.50, 8.20 (11.89 max)
  Total Memory          : 16384M
  Kernel Uptime         : 0D 0H 49M
  Last Reading:
    CPU Usage           : 16.6% user, 10.5% sys, 0.0% io, 4.6% irq, 68.3% idle
    Poll CPUs           : 5 (1, 2, 3, 4, 5)
    Processes / Tasks   : 234 processes / 54 tasks
    Network             : 353.452 kpps rx, 3612.279 mbps rx, 282.869 kpps tx, 2632.760 mbps
 tx
    File Usage          : 2336 open files, 1631523 available
    Memory Usage        : 4280M 26.1% used, 42M 0.3% reclaimable
  Maximum/Minimum:
    CPU Usage           : 23.2% user, 11.2% sys, 0.1% io, 5.5% irq, 61.5% idle
    Poll CPUs           : 5 (1, 2, 3, 4, 5)
    Processes / Tasks   : 244 processes / 54 tasks
    Network             : 453.449 kpps rx, 4635.918 mbps rx, 368.252 kpps tx, 3483.816 mbps
```

```
tx
   File Usage          : 3104 open files, 1630755 available
   Memory Usage        : 4318M 26.4% used, 46M 0.3% reclaimable
```

Capacity, CEPS and Throughput

Sizing a VPC-SI instance requires modeling of the expected call model.

Many service types require more resources than others. Packet size, throughput per session, CEPS (Call Events per Second) rate, IPSec usage (site-to-site, subscriber, LI), contention with other VMs, and the underlying hardware type (CPU speed, number of vCPUs) will further limit the effective number of maximum subscribers. Qualification of a call model on equivalent hardware and hypervisor configuration is required.

Diagnostics and Monitoring

Because VPC-SI runs within VMs, no hardware diagnostics or monitoring are provided. Retrieval of hardware sensor data (temperature, voltage, memory errors) are accomplished via the hypervisor and external monitoring systems.

VPC-SI monitors and exports vCPU, vRAM, and vNIC usage through existing mechanisms including CLI **show** commands, bulkstats and MIB traps. However, an operator may find that monitoring physical CPU, RAM, and NIC values in the hypervisor is more useful.

Because vNICs have a variable max throughput (not defined as 1 Gbps or 10 Gbps for example), counters and bulkstats that export utilization as a percentage of throughput may have little value. Absolute values (bps) can be obtained from VPC-SI, but where possible physical infrastructure utilization should be obtained from the hypervisor. This would not apply to pass-through PF NICs, as those will have a fixed maximum throughput.

StarOS VPC-SI Build Components

The following StarOS build filename types are associated with VPC-SI:

- **qvpc-si-<version>.iso** initial installation or startover ISO file.
- **qvpc-si-<version>.bin** update, upgrade or recovery file for a system that is already running. For additional information refer to the *StarOS Management Operations* chapter.
- **qvpc-si-template-libvirt-kvm-<version>.tgz** KVM libvirt template plus ssi_install.sh.
- **qvpc-si-template-vmware-<version>.ova** VMware OVA template.
- **qvpc-si-<version>.qcow2.gz** KVM QCOW2 disk template.

VPC-SI Boot Parameters

The boot parameters file provides a means to pass configuration items to StarOS before it boots. These parameters specify items such as the management, service, and VNFM interface details, as well as configuration for the Internal Forwarder Task (iftask) created at StarOS start up.

The boot parameters are sourced in multiple ways, with all methods using the same parameter names and usage. The first location for the boot parameters file is on the first partition of the first VM drive, for example, */boot1/param.cfg*. The second location searched is on the configuration drive, which is a virtual CD-ROM drive. If you are using OpenStack, specify the target boot parameters file name as *staros_param.cfg*. If you are not using OpenStack, create an ISO image with *staros_param.cfg* in the root directory and attach this ISO to the first virtual CD-ROM drive of the VM.

As the VM boots, the param.cfg file is parsed first by the preboot environment known as CFE. Once the VM starts Linux, the virtual CD-ROM drive is accessed to parse the staros_param.cfg file. If there are any conflicts with values stored in the */boot1/param.cfg* file, parameters in staros_param.cfg take precedence.

Format of the Boot Parameters File

The structure of the boot parameters file is:

VARIABLE_NAME = VALUE

Specify one variable per line with a newline as the end of the line terminator (UNIX text file format). Variable names and values are case insensitive. Invalid values are ignored and an error indication is displayed on the VM console. If there are duplicate values for a variable (two different values specified for the same variable name), the last value defined is used.

Numeric values do not need to be zero padded. For example a PCI_ID of 0:1:1.0 is treated the same as 0000:01:01.0.

Network Interface Identification

VPC-SI assigns the vNIC interfaces in the order offered by the hypervisor. You cannot be guaranteed that the order of the vNICs as listed in the hypervisor CLI/GUI is the same as how the hypervisor offers them to the VM.

The order that VPC-SI finds the vNICs is subject to the PCI bus enumeration order and even paravirtual devices are represented on the PCI bus. The PCI bus is enumerated in a depth first manner where bridges are explored before additional devices at the same level. If all the network interfaces are of the same type then knowing the PCI topology is sufficient to get the vNIC order correct. If the network interfaces are of different types, then the order is dependent on the PCI topology plus the device driver load order inside the VM. The device driver load order is not guaranteed to be the same from software release to release but in general paravirtual devices are prior to pass-through devices.

There are several methods available to identify NICs.

- MAC address: MAC address of the interface
- Virtual PCI ID

- Bonded interfaces: When using network device bonding, network interfaces are identified to serve as the slave interface role. The slave interfaces in the bond are identified using MAC, PCI ID, or Interface type.
- Interface type and instance number.

Virtual PCI ID

Devices on a PCI bus are identified by a unique tuple known as the domain, bus, device, and function numbers. These identifiers can be identified in several ways.

Inside the guest, the **lspci** utility shows the bus configuration:

```
# 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 System peripheral: Intel Corporation 6300ESB Watchdog Timer
00:04.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
00:05.0 Ethernet controller: Red Hat, Inc Virtio network device
00:06.0 Ethernet controller: Red Hat, Inc Virtio network device
```

The domain, bus, device, and function numbers for this virtual bus are shown here:

Table 2: Virtual PCI IDs

Line	Domain	Bus	Device	Function
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)	0	0	0	0
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]	0	0	1	0
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]	0	0	1	1
00:01.2 USB controller: Intel Corporation 82371SB PIIX3 USB [Natoma/Triton II] (rev 01)	0	0	1	2
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)	0	0	1	3
00:02.0 VGA compatible controller: Cirrus Logic GD 5446	0	0	2	0
00:03.0 System peripheral: Intel Corporation 6300ESB Watchdog Timer	0	0	3	0
00:04.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon	0	0	4	0
00:05.0 Ethernet controller: Red Hat, Inc Virtio network device	0	0	5	0

Line	Domain	Bus	Device	Function
00:06.0 Ethernet controller: Red Hat, Inc Virtio network device	0	0	6	0

For libvirt-based virtual machines, you can get the virtual PCI bus topology from the **virsh dumpxml** command. Note that the libvirt schema uses the term *slot* for the device number. This is a snippet of the xml description of the virtual machine used in the previous example:

```
<interface type='bridge'>
  <mac address='52:54:00:c2:d0:5f'/>
  <source bridge='br3043'/>
  <target dev='vnet0'/>
  <model type='virtio'/>
  <driver name='vhost' queues='8'/>
  <alias name='net0'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
</interface>
<interface type='bridge'>
  <mac address='52:54:00:c3:60:eb'/>
  <source bridge='br0'/>
  <target dev='vnet1'/>
  <model type='virtio'/>
  <alias name='net1'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
</interface>
```

Interface Type and Instance Number

Here the NIC is identified by its type using its Linux device driver name (virtio_net, vmxnet3, ixgbe, i40e, etc) and its instance number. The instance number is based on PCI enumeration order for that type of interface starting at instance number 1. The interface type is available to identify both paravirtual types as well as pass-through interfaces and SR-IOV virtual functions. The PCI enumeration order of devices on the PCI bus can be seen from the **lspci** utility.

For example, a CF with the following guest PCI topology indicates that virtio_net interface number1 is the Ethernet controller at 00:05.0 and virtio_net interface number 2 is the Ethernet Controller at 00:06.0. The output is from the **lspci** command executed in the guest:

```
# 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 System peripheral: Intel Corporation 6300ESB Watchdog Timer
00:04.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
00:05.0 Ethernet controller: Red Hat, Inc Virtio network device
00:06.0 Ethernet controller: Red Hat, Inc Virtio network device
```

Here is the complete list of the supported Linux drivers:

Table 3: Supported Linux Drivers

Type	PCI Vendor / Device ID	Driver Name
VIRTIO (paravirtual NIC for KVM)	0x10af / 0x1000	virtio_net
VMXNET3 (paravirtual NIC for VMware)	0x15ad / 0x07b0	vmxnet3
Intel 10 Gigabit Ethernet	0x8086 / 0x10b6 0x8086 / 0x10c6 0x8086 / 0x10c7 0x8086 / 0x10c8 0x8086 / 0x150b 0x8086 / 0x10dd 0x8086 / 0x10ec 0x8086 / 0x10f1 0x8086 / 0x10e1 0x8086 / 0x10db 0x8086 / 0x1508 0x8086 / 0x10f7 0x8086 / 0x10fc 0x8086 / 0x1517 0x8086 / 0x10fb 0x8086 / 0x1507 0x8086 / 0x1514 0x8086 / 0x10f9 0x8086 / 0x152a 0x8086 / 0x1529 0x8086 / 0x151c 0x8086 / 0x10f8 0x8086 / 0x1528 0x8086 / 0x154d 0x8086 / 0x154f 0x8086 / 0x1557	ixgbe
Intel 10 Gigabit NIC virtual function	0x8086 / 0x10ed 0x8086 / 0x1515	ixgbevf

Type	PCI Vendor / Device ID	Driver Name
Cisco UCS NIC	0x1137 / 0x0043 0x1137 / 0x0044 0x1137 / 0x0071	enic
Mellanox ConnectX-5 **Note** Mellanox is supported only on the User Plane.	0x15b3 / 0x1017 0x15b3 / 0x1018	mlx5_core
Intel 710 family NIC (PF)	0x8086 / 0x1572 (40 gig) 0x8086 / 0x1574 (40 gig) 0x8086 / 0x1580 (40 gig) 0x8086 / 0x1581 (40 gig) 0x8086 / 0x1583 (40 gig) 0x8086 / 0x1584 (40 gig) 0x8086 / 0x1585 (40 gig) 0x8086 / 0x158a (25 gig) 0x8086 / 0x158b (25 gig)	i40e**
Intel 710 family NIC virtual function	0x8086 / 0x154c	i40evf

** Note: A known issue exists where MAC address assignment does not occur dynamically for SRIOV VFs created on the host when using the **i40e** driver. MAC address assignment is necessary to boot the StarOS VM. As a workaround, MAC address assignment must be configured from the host. Refer to the following link for more information:https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/xl710-sr-iov-config-guide-gbe-linux-brief.pdf

Configuring VPC-SI Boot Parameters

VPC-SI VMs have one interface configured to connect to the management network. This interface is typically configured in StarOS and should be part of the Day 0 configuration. The management interface supports static address assignment through the main StarOS configuration file.

An additional 0 to 4 network interfaces serve as service ports. These interfaces are configured by StarOS. Typically these ports are configured as trunk ports in the VNF infrastructure (VNFI).

VPC-SI VMs have the option of having a network interface that is connected to the virtual network function (VNF) manager (VNFM) if it exists. This interface can be configured via DHCP or static IP assignment and is used to talk to a VNFM or higher level orchestrator. This interface is enabled before the main application starts.

Table 4: VPC-SI Boot Parameters

Parameter	Description
MGMT_INTERFACE=*value*	Interface to the management port on the VPC-SI VM. Valid values are: • MAC: xx:xx:xx:xx:xx:xx • PCI_ID: xxx:xx:xx.x (Domain:Bus:Device.Function) • TYPE: *drive-name-instance-number* • BOND: *slave-interface-A,slave-interface-B* Refer to Network Interface Identification, on page 12 for information on determining the interface identifier.
SERVICE#_INTERFACE=*value*	Interface to a service port number #, where # can be from 1 to 4. Service interfaces support the same values listed above for the **MGMT_INTERFACE**.
SERVICE_INTERFACE_MTU=*value*	By default, the IFTASK process sets the maximum interface MTU as 2100 bytes. *value* must be an integer from 576 to 9100 bytes.
VNFM_INTERFACE=*value*	Optional network interface to the Virtual Network Function Manager (VNFM) or orchestrator. VNFM interfaces support the same values listed above for the **MGMT_INTERFACE**.
VNFM_IPV4_ENABLE=*{true | false }*	Enables the VNFM interface.
VNFM_IPV4_DHCP_ENABLE=*{true | false }*	Enables DHCP to the VNFM.
VNFM_IPV4_ADDRESS=*value*	Specifies the IP address for the VNFM where DHCP is not used.
VNFM_IPV4_NETMASK=*value*	Specifies the netmask for the IP address of the VNFM where DHCP is not used.
VNFM_IPV4_GATEWAY=*value*	Specifies the gateway for the IP address of the VNFM where DHCP is not used.

Parameter	Description
IFTASK_CRYPTO_CORES=*value*	(Optional) (Optional) IFTASK_CRYPTO_CORES When IFTASK_SERVICE_TYPE is configured to "2" (EPDG), this parameter specifies the percentages of iftask cores to allocate to crypto processing. Values can range from 0 to 50 percent, though the cores dedicate will be capped at 4. The default is 0. This parameter should only be used if the IFTASK_SERVICE_TYPE is set to "2" (EPDG). If it is set to any other service type, then this parameter should be set to "0".
IFTASK_MCDMA_CORES=*value*	(Optional) Sets the number of cores assigned to Multi-channel Direct Memory Access (MCDMA) to be a percentage of total iftask cores. You must first define IFTASK_CORES parameter above. NOTE: When NUMA optimization is enabled and also this MCDMA core count setting is configured you must set IFTASK_CORES=percentage-of-cores to be an even number. This ensures that the MCDMA threads are evenly distributed across the NUMA nodes.
CHASSIS_ID=*value*	The chassis ID protects select commands in the CLI configuration such as administrator credentials, snmp community, radius/diameter data, and the authentication data. It is required to save or load configuration and is normally generated using random data and a user seed from a CLI command. The chassis ID is then saved on the “/flash” storage of the system to be used later to load saved configurations. Configuring this option loads the the chassis ID from the parameters configuration files from /flash. The chassis id on /flash will be ignored (i.e. support upgrade from previous version) and generation of new chassis id using the CLI commands will be prevented. The chassis ID will no longer be stored on /flash if provided using this method. This will also require ESC (or another VNFM) to manage and provide the chassis id to the VM.

Parameter	Description
IFTASK_SERVICE_TYPE=*value*	(Optional) Specifies the service type being deployed in order to calculate the service memory and enable service-specific features. The default is 0. The following service types can be specified: • 0 = VPC service type • 1 = GiLAN service type • 2 = ePDG service type • 3 = CUPS controller service type • 4 = CUPS forwarder service type
FORWARDER_TYPE=*value*	Specifies the forwarder type as "vpp" or "iftask". By default, the forwarder type is "iftask". The forwarder type is VPP only when the FORWARDER_TYPE is properly configured as VPP. For example, **FORWARDER_TYPE=***vpp*

Example params.cfg

```
MGMT_INTERFACE=MAC:00:01:02:03:04:05
SERVICE1_INTERFACE=PCI_ID:0000:01:02.0
SERVICE2_INTERFACE=PCI_ID:0000:01:03.0
VNFM_INTERFACE=MAC:00:01:02:03:04:AA
VNFM_IPV4_ENABLE=true
VNFM_IPV4_DHCP_ENABLE=false
VNFM_IPV4_ADDRESS=10.1.1.100
VNFM_IPV4_NETMASK=255.255.255.0
VNFM_IPV4_GATEWAY=10.1.1.1
IFTASK_MCDMA_CORES=20
CHASSIS_ID=2sg9x1wqbj
IFTASK_SERVICE_TYPE=0
```

Use the StarOS command **show cloud hardware iftask** to verify that the iftask boot parameters took effect.

Configuring VNFM Interface Options

Note These configuration options are optional.

The virtual network functions manager (VNFM) interface is designed to communicate between each VM and a VNFM. The VNFM interface initializes before the main application and only boot parameters can configure the interface.

The VNFM interface is disabled by default.

Enable VNFM IPv4 Interface

The default value is False (disabled).

Variable	Valid Values
VNFM_IPV4_ENABLE	True or False

Configure IPv4 DHCP Client

Variable	Valid Values
VNFM_IPV4_DHCP_ENABLE	True or False

Configure IPv4 Static IP

Note If IPv4 DHCP client is enabled, static configuration parameters are ignored.

Variable	Valid Values
VNFM_IPV4_ADDRESS	x.x.x.x
VNFM_IPV4_NETMASK	x.x.x.x
VNFM_IPV4_GATEWAY	x.x.x.x

Enable VNFM IPv6 Interface.

Variable	Valid Values
VNFM_IPV6_ENABLE	True or False

Enable IPv6 Static IP Configuration

Variable	Valid Values
VNFM_IPV6_STATIC_ENABLE	True or False

If set to true, static IP parameters configuration applies to the interface as shown in the following section. If set to false, the interface attempts to use both stateless autoconfiguration (RFC4862) and DHCPv6 to configure the address of the interface.

Configure IPv6 Static IP

Note If the "VNFM_IPV6_ENABLE" parameter value is set to false, the static configuration parameters are ignored. The IPv6 address field should conform to RFC 5952. Prefix is fixed at /64.

Variable	Valid Values
VNFM_IPV6_ADDRESS	x:x:x:x:x:x:x:x
VNFM_IPV6_GATEWAY	x:x:x:x:x:x:x:x

VPP Configuration Parameters

The following sections list the parameters that are applicable only when the FORWARDER_TYPE selected is VPP. These parameters enable a fine-grained control over the CPU of the VPP and the interface configuration.

Note Before overriding any of the VPP configuration parameters, ensure that you contact your Cisco account representative to help identify the override values.

VPP CPU Assignment

VPP workers are real-time threads that consume an entire CPU core. While the VPP main thread does not consume an entire core, it can be busy. Therefore, assign it to avoid conflicts.

The following table lists the VPP-CPU parameters.

Table 5: VPP-CPU Parameters

Parameter	Description
VPP_CPU_MAIN=*value*	Specifies the Linux processor number, that is 0 – (number of CPUs minus 1). The default value is 1. Use the following example to set the main thread value to the Linux processor number 1. `VPP_CPU_MAIN=1`
VPP_CPU_WORKER_CNT=*value*	Specifies the number of worker threads set on the Linux processor. The valid value is 0 – (number of CPUs minus 3). The default value is 50% of the Linux CPUs or number of CPUs in the VPP_CPU_WORKER_LIST. Use the following example to set the number of worker threads to 3. `VPP_CPU_WORKER_CNT=3`

Parameter	Description
VPP_CPU_WORKER_LIST=*value*	Specifies the worker threads set on the Linux processor. The worker list is a comma-separated list of Linux processor numbers. The valid value is 0 – (number of CPUs minus 1). The default is a round-robin number that is assigned across all sockets, skipping the first core on all sockets and the second core on the first socket. Use the following example to set the number of worker threads to Linux processors 2, 9, and 10 `VPP_CPU_WORKER_LIST=2,9,10`

Default DPDK Configuration

The following parameters that are listed in the table configure DPDK in general or set the interface defaults.

Table 6: DPDK Parameters

Parameter	Description
VPP_DPDK_BUFFERS=*value*	Specifies the number of DPDK buffers. The minimum buffer is 32,000 and the maximum is based on the VM size. The default number of buffers is 128,000. Use the following example to set the DPDK buffer to 200,000. `VPP_DPDK_BUFFERS=200000`
VPP_DPDK_RX_QUEUES=*value*	Specifies the number of RX queues for all interfaces that do not have a specific configuration. The valid values range from 1 through 64 depending on the interface type and host configuration. The default value is calculated from the VPP_CPU_WORKER_COUNT to minimize the overall number of queues while having at least one queue assigned to each worker. Use the following example to set the default number of RX queues to 2. **Note** Values that do not match can be set based on the interface type and host configuration. `VPP_DPDK_RX_QUEUES=2`

Parameter	Description
VPP_DPDK_TX_QUEUES=*value*	Specifies the number of TX queues for all interfaces that do not have a specific configuration. The valid values range from 1 through 64 depending on the interface type and host configuration. The default value is VPP_DPDK_RX_QUEUES. Use the following example to set the default number of TX queues to 4. **Note** Values that do not match can be set based on the interface type and host configuration. `VPP_DPDK_TX_QUEUES=4`
VPP_DPDK_RX_DESCS=*value*	Specifies the number of RX descriptors for all interfaces that do not have a specific configuration. The valid values range from 128 through 128,000 depending on the interface type and host configuration. The default value is an unspecified driver-dependent value. Use the following example to set the default number of RX descriptors to 256. **Note** Values that do not match can be set based on the interface type and host configuration. `VPP_DPDK_RX_DESCS=256`
VPP_DPDK_TX_DESCS=*value*	Specifies the number of TX descriptors for all interfaces that do not have a specific configuration. The valid values range from 128 through 128,000 depending on the interface type and host configuration. The default value is VPP_DPDK_RX_DESCS or an unspecified driver-dependent value. Use the following example to set the default number of TX descriptors to 512. **Note** Values that do not match can be set based on the interface type and host configuration. `VPP_DPDK_TX_DESCS=512`

Interface-Specific Configuration

The following parameters that are listed in the table refine individual interfaces. These parameters also override the default DPDK configuration, where applicable.

Table 7: Interface Parameters

Parameter	Description
<ROLE>_VPP_RX_QUEUES=*value*	Specifies the number of RX queues for the interface ROLE. The valid values range from 1 through 64 depending on the interface type and host configuration. The default value is unspecified. Use the following example to set the number of RX queues for service port 1–2. `SERVICE1_INTERFACE_VPP_RX_QUEUES=2`
<ROLE>_VPP_TX_QUEUES=*value*	Specifies the number of TX queues for the interface ROLE. The valid values range from 1 through 64 depending on the interface type and host configuration. The default value is <ROLE>_VPP_RX_QUEUES or is unspecified.
<ROLE>_VPP_RX_DESCS=*value*	Specifies the number of RX descriptors for the interface ROLE. The valid values range from 128 through 128,000 depending on the interface type and host configuration. The default value is unspecified. Use the following example to set the number of RX descriptors for service port 1 to 1024. `SERVICE1_INTERFACE_VPP_RX_DESCS=1024`
<ROLE>_VPP_TX_DESCS=*value*	Specifies the number of RX descriptors for the interface ROLE. The valid values range from 128 through 128,000 depending on the interface type and host configuration. The default value is <ROLE>_VPP_RX_DESCS or is unspecified.
<ROLE>_VPP_WORKER_LIST=*value*	Specifies the worker threads for the interface ROLE. The worker list is a comma-separated list of Linux processor numbers. The valid value is 0 – (number of CPUs minus 1). The default is a round-robin number that is assigned across all sockets, skipping the first core on all sockets and the second core on the first socket. Use the following example to set the worker thread list for service port 1 to Linux processors 2 and 3. `SERVICE1_INTERFACE_VPP_WORKER_LIST=2,3`

Software Installation and Network Deployment

This guide assumes that VPC-SI has been properly installed to run in a virtual machine (VM) on a commercial off-the shelf (COTS) server.

For additional information on supported operating system and hypervisor packages, as well as platform configurations, please contact your Cisco representative. The Cisco Advanced Services (AS) group offer consultation, installation and network deployment services for the VPC-SI product.