Linux on IBM Z and LinuxONE

Networking with RoCE Express
May 2022

SC34-7745-00

Note

Before using this information and the product it supports, read the information in "Notices" on page 39.

This edition applies to the Linux on IBM Z and LinuxONE Development stream in May 2022 until otherwise indicated in new editions.

Contents

About this publication

This publication explores what RoCE Express adapters offer for network connections of Linux® on IBM Z® and IBM® LinuxONE.

The publication applies to RoCE Express2 or later RoCE Express adapters. The descriptions in this publication reflect the available technologies at the time of writing.

Unless stated otherwise, all IBM z/VM® related information in this document assumes a current z/VM version, see www.vm.ibm.com/techinfo/lpmigr/vmleos.html.

You can find the latest version of this and other publications in the Linux on IBM Z and LinuxONE library on IBM Documentation at ibm.com/docs/en/linux-on-systems?topic=linuxone-library-overview.

Where to get more information

Find more information about RoCE Express and the Linux on IBM Z and LinuxONE environment.

For more information about the device drivers for Linux on IBM Z, see *Device Drivers, Features, and Commands*. You can obtain the latest version of this publication on IBM Documentation at ibm.com/docs/en/linux-on-systems?topic=overview-device-drivers-features-commands.

For information about KVM on IBM Z and LinuxONE, see *KVM Virtual Server Management*, SC34-2752. You can obtain the latest version of this publication on IBM Documentation at ibm.com/docs/en/linux-on-systems?topic=overview-kvm-virtual-server-management.

For information about z/VM, see the documentation for your z/VM version on IBM Documentation at ibm.com/docs/en/zvm or see the z/VM PDF library at www.ibm.com/vm/library/index.html.

Other publications for Linux on IBM Z and LinuxONE

You can find publications for Linux on IBM Z and LinuxONE on IBM Documentation.

These publications are available on IBM Documentation at ibm.com/docs/en/linux-on-systems?topic=linuxone-library-overview

- *Device Drivers, Features, and Commands*
- *Using the Dump Tools*
- *How to use FC-attached SCSI devices with Linux on z Systems®*, SC33-8413
- *Networking with RoCE Express*, SC34-7745
- *KVM Virtual Server Management*, SC34-2752
- *Configuring Crypto Express Adapters for KVM Guests*, SC34-7717
- *Introducing IBM Secure Execution for Linux*, SC34-7721
- *openCryptoki - An Open Source Implementation of PKCS #11*, SC34-7730
- *libica Programmer's Reference*, SC34-2602
- *libzpc - A Protected-Key Cryptographic Library*, SC34-7731
- *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294
- *Pervasive Encryption for Data Volumes*, SC34-2782
- *Enterprise Key Management for Pervasive Encryption of Data Volumes*, SC34-7740
- *How to set an AES master key*, SC34-7712

- *Troubleshooting*, SC34-2612
- *Kernel Messages*, SC34-2599
- *How to Improve Performance with PAV*, SC33-8414
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596

Chapter 1. Introduction to RoCE Express

RoCE Express is a PCI network adapter for IBM Z and LinuxONE hardware. You can order RoCE Express as a hardware feature.

Figure 1. 10 GbE RoCE Express2 feature with two adapter ports

About RoCE Express

Depending on your environment and requirements, RoCE Express can be an alternative to network adapters that are attached through the IBM Z channel subsystem.

Why use RoCE Express

Managing RoCE Express on Linux on IBM Z is similar to managing RoCE-capable PCI-based network interfaces on Linux for any hardware platform, largely using the same tools. With RoCE Express, you are in line with networking in the distributed world and set to benefit from developments in that area.

RoCE Express has low latency and depending on your workload, can offer performance advantages.

Apart from TCP/IP connections, RoCE Express supports RDMA protocols, including Shared Memory Communications Remote (SMC-R) connections, which tend to be advantageous for long-lasting connections with large data transfers, see “Modes of operation” on page 2.

RoCE Express provides network interfaces that benefit from the hardware environment and offer state-of-the-art networking capabilities, see “RoCE Express capabilities” on page 3.

For a comparison of RoCE Express with other adapter options, see *Exploring the performance of network adapters for Linux on IBM Z* [PDF].

Physical functions, virtual functions, and ports

For RoCE Express2 and later, each physical adapter port maps to one physical function (PF). Physical functions are used for hardware-related management and for special, sensitive tasks like firmware updates.

Depending on the physical slot at which the RoCE Express adapter is plugged, the adapter is managed through one of the PCI resource groups.

Each physical function has multiple virtual functions (VFs). The hardware administrator can make these virtual functions eligible for LPARs or DPM partitions. Virtual functions belong to the same resource group as their associated physical function.

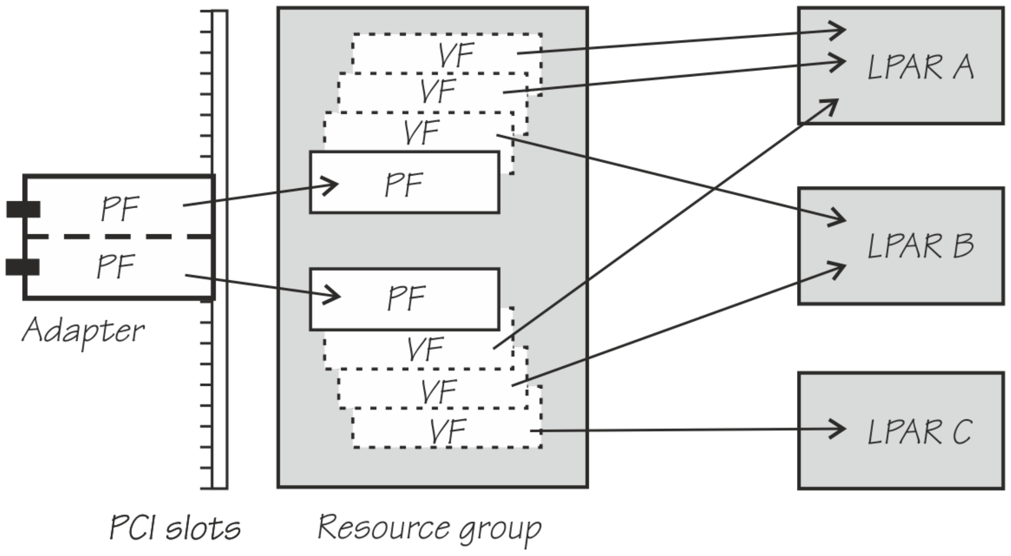


Figure 2. 10 GbE RoCE Express2 feature with two adapter ports

Only virtual functions can be used directly in Linux on IBM Z. Physical functions are hidden in Linux on IBM Z, so this publication uses the term *PCI function* to mean virtual function.

The maximum number of virtual functions for a physical function depends on the model of the RoCE Express feature.

Table 1. Ports, physical functions, and virtual functions of RoCE Express features

Feature	Physical adapter ports	Ports per PF	Maximum VFs per PF
RoCE Express3	2	1	63
RoCE Express2.1 and RoCE Express2	2	1	63
RoCE Express (see Note)	2	2	31

Note: RoCE Express is included in Table 1 on page 2 for completeness. This publication applies to RoCE Express2 or later.

As of RoCE Express2, all PCI functions that are associated with the same physical function share one of the physical adapter ports.

Modes of operation

RoCE Express PCI functions have modes for two connection types: IP-based connections and RDMA-based connections.
Both connection types can be used concurrently on the same PCI function. The connection type depends on the socket application and how it is started, see "Using a PCI function for SMC-R connections" on page 19.

The adapter port of an operational PCI function must be connected to an external LAN. Even so, for both connection types communication peers on the same hardware system that share a RoCE Express adapter port can communicate through the adapter without routing packets through the LAN.

TCP/IP connections

You can set up connections to any communication peer that you can reach with TCP/IP or other IP-based protocols. The network adapter of the communication peer can be RoCE Express, but it can also be a different adapter.

In particular, possible peers include z/OS®, z/VSE®, and Linux instances that are reached through an OSA adapter.

SMC-R connections

Shared Memory Communications Remote (SMC-R) uses remote direct memory access (RDMA) over Converged Ethernet (RoCE) for high-throughput, low-latency connections.

Both communication peers must support SMC-R connections.

A regular TCP/IP connection between the peers must be in place to initiate an SMC-R connection. On both sides of the connection, the PCI function must be associated with the TCP/IP network device, see "Using a PCI function for SMC-R connections" on page 19.

Linux on IBM Z can use SMC-R connections to instances of Linux or of z/OS. On Linux, the initial TCP/IP connection can use any available network adapter, including the RoCE Express adapter that is also used for the SMC-R connection. In contrast, a z/OS peer needs an OSA adapter for the TCP/IP connection.

Extra handshaking in the protocol incurs more initial latency for establishing an SMC-R connection than for opening a TCP/IP connection. Once established, the SMC-R connection can combine superior data throughput with low latency and reduced CPU consumption. Expect best results for multiple concurrent and long-lasting connections for transferring large amounts of data.

z/OS connectivity

z/OS requires an OSA-Express adapter to connect to an external LAN.

z/OS can use RoCE Express for RDMA traffic as used by SMC-R connections, but not for TCP/IP connections.

Because SMC-R connections are initialized through a TCP/IP connection, z/OS needs an OSA-Express adapter to establish SMC-R connections through RoCE Express.

At the Linux end of a TCP/IP connection, you can use an OSA-Express adapter or a RoCE Express adapter. You can connect a RoCE Express adapter from Linux to an OSA-Express adapter at the z/OS end of the connection through a LAN switch, a router, or even through direct cabling.

RoCE Express capabilities

PCI functions of RoCE Express support most of the typical network interface capabilities. The following list provides selected examples, without any claim to completeness.

Hardware offloads
By default, the PCI functions use hardware acceleration, for example, for checksums and for TCP segmentation.

HSCI interfaces
You can connect a PCI function with a HiperSockets device to create a HiperSockets Converged Interface (HSCI). For more information about HSCI, see *Device Drivers, Features, and Commands*.

rdma-core library
The PCI functions support the rdma-core library.

iSER
The PCI functions support the iSCSI Extensions for RDMA (iSER).

NVMe
The PCI functions support NVMe over RDMA.

VLAN
The PCI functions support VLAN configurations.

QoS
You can prioritize network traffic to attain quality of service (QoS) goals.

ECN and flow control
The PCI functions support explicit congestion notification (ECN) and flow control.

Jumbo frames
The PCI functions support jumbo frames.

You can use the **`ethtool`** and **`ip`** commands for a transient configuration of your network interface. For a persistent configuration, use the network configuration tools of your distribution.

RoCE Express limitations

At the time of writing, RoCE Express does not offer all capabilities of OSA-Express.

Promiscuous mode
Linux instances do not have access to the physical functions of RoCE Express. As a consequence, capabilities that require a physical function are not available. In particular, you cannot run network interfaces of PCI functions in promiscuous mode.

z/VM VSWITCH support
You cannot add network interfaces of PCI functions to a z/VM VSWITCH.

z/OS connectivity
z/OS cannot use RoCE Express for TCP/IP connections, see also "z/OS connectivity" on page 3.

Management and configuration tools

Use the `smc-tools` package, distribution tools, and generic PCI management commands to manage and configure RoCE Express PCI functions.

smc-tools package

IBM provides the smc-tools package to manage SMC-related PCI functions and devices on Linux on IBM Z and LinuxONE, including the PCI functions of RoCE Express.

Install the package that is included in your distribution. If you distribution does not include the package, you can obtain the latest version from GitHub at github.com/ibm-s390-linux/smc-tools/tags.

PCI tools

You can use common PCI tools like **`lspci`** to work with RoCE Express PCI functions.

IBM provides the **`zpcictl`** command for handling defective PCI functions on IBM Z. Depending on your distribution, the package that includes this command might be called `s390-tools` or `s390utils`.

Network configuration

You can use common network management tools like **`ip`** and **`ethtool`** for exploring your network and network interfaces.

To persistently configure your network, use the network management tool of your distribution. For example, use NetworkManager for Red Hat® Enterprise Linux 8, use wicket for SUSE Linux Enterprise Server.

Models and supported environments

You can use the RoCE Express features of your IBM Z or LinuxONE hardware for network attachment of Linux in traditional LPAR mode, in DPM partition mode, or as a KVM or z/VM guest.

Available models and hardware support

Support for RoCE Express features depends on your IBM Z or LinuxONE hardware.

The following table lists the available RoCE Express features, as of RoCE Express2, for each hardware model. The full feature names have prefixes "10 GbE" for 10-Gigabit Ethernet and "25 GbE" for 25-Gigabit Ethernet.

Table 2. Support for RoCE Express features

Feature	**IBM z16**	**z15™ and LinuxONE III**	**z14 and LinuxONE II**
RoCE Express3	25 GbE 10 GbE	Not supported	Not supported
RoCE Express2.1	25 GbE 10 GbE	25 GbE 10 GbE	Not supported
RoCE Express2	25 GbE 10 GbE	25 GbE 10 GbE	25 GbE 10 GbE

All RoCE Express features use optical connectors.

Switch requirements for the 25 GbE features:

- You need 25-Gigabit Ethernet capable switches. A fallback to 10-Gigabit Ethernet is not supported.
- For connections to a Cisco switch, you must set the switch to use "FEC RS-IEEE" for forward-error-correction (FEC) in Reed-Solomon mode.

Hypervisor support

You can use PCI functions of RoCE Express with Linux in LPAR or DPM partition mode or as a KVM or z/VM guest.

Working with PCI functions is similar regardless of how Linux is hosted. For example, you can configure for high availability through bonded interfaces on Linux in all environments.

Notes:

- Other than OSA Express, RoCE Express does not support z/VM VSWITCH technology to provide path redundancy.
- Because RoCE Express does not provide a promiscuous mode, you cannot use Open vSwitch in a KVM host to provide path redundancy for its guests.

Distribution support

The major distributions for Linux on IBM Z support RoCE Express. On the distribution, you need the `mlx5` device driver.

The following table shows the minimum level for the major distributions for Linux on IBM Z.

Table 3. Distribution support for RoCE Express features

Feature	**Red Hat Enterprise Linux**	**SUSE Linux Enterprise Server**	**Ubuntu Server**
RoCE Express3	As of 8.4 at the latest service level	As of 15 SP3 at the latest service level	As of 20.04 LTS at the latest service level
RoCE Express2.1 and RoCE Express2	As of 7.3 at the latest service level	As of 12 SP3 at the latest service level	As of 16.04 LTS at the latest service level

Identifiers

You must know how to identify PCI functions within Linux and on hardware interfaces.

Identifiers for the PCI function

On IBM Z, PCI functions are identified by general PCI identifiers, but also by identifiers that are specific to z/Architecture®.

Identifiers specific to z/Architecture

The following identifiers are specific to z/Architecture hardware systems.

FID

Function IDs (FIDs) are 8-digit hexadecimal values that uniquely identify PCI functions within the scope of the IBM Z or LinuxONE hardware system. Many interfaces drop leading zeros from FIDs.

FIDs are assigned in the hardware configuration and persist across PORs. A KVM or z/VM hypervisor can map the FID of the hardware to a different FID in a guest.

To set a PCI function online or offline in Linux, you must know its FID.

UID

User defined identifiers (UIDs) are four-digit hexadecimal values that identify PCI functions within the scope of an LPAR or DPM partition.

The hardware administrator can assign UIDs in the hardware configuration. The hardware configuration of the partition must enforce UID uniqueness to rule out multiple PCI functions with the same UID within the partition.

UID uniqueness is always enforced in DPM partitions. For traditional LPARs, the hardware administrator must set the **UID uniqueness** option in the LPAR configuration. For more details, see "UID uniqueness" on page 8.

PCHID

The physical channel identifier (PCHID) is the physical address of a channel path in the hardware. All PCI functions of a RoCE Express adapter have the same PCHID.

Generic PCI identifiers

The following identifiers are commonly used for PCI devices, across various hardware platforms.

PCI ID

The PCI ID is the function address that identifies the PCI function in the PCI stack. For Linux on IBM Z, the PCI ID has this format *<UID>*:00:*<device>*.*<function>*. All elements of the function address represent aspects of the logical PCI topology. The *<device>*.*<function>* part is usually `00.0`. The **`lspci`** command lists PCI devices by PCI ID.

IB device name

The name that the InfiniBand (IB) device driver uses for the PCI function. Because RDMA is based on the IB communications standard, some terms that are used in the RDMA context refer to IB.

Depending on your environment, the UID, FID, or both, can be used in the naming scheme of your network interfaces, see "Network interface names" on page 8.

Mapping the identifiers in Linux

Use the **smc_rnics** command to list PCI functions with their identifiers. By default, the command shows the network device information.

```
# smc_rnics
  FID Power PCI_ID       PCHID Type          PPrt PNET_ID Net-Dev
-------------------------------------------------------------------
  8ca 1     0008:00:00.0 01c8  RoCE_Express2 0    NET25   eno8
  8ea 1     0009:00:00.0 01c8  RoCE_Express2 1    NET26   eno9
```

Use the **-I** option to display an InfiniBand view of this list, with RDMA information.

```
# smc_rnics -I
  FID Power PCI_ID       PCHID Type          IPrt PNET_ID IB-Dev
-------------------------------------------------------------------
  8ca 1     0008:00:00.0 01c8  RoCE_Express2 1    NET25   mlx5_1
  8ea 1     0009:00:00.0 01c8  RoCE_Express2 1    NET26   mlx5_2
```

As shown in the example output, the default network-device properties and RDMA properties have two diverging columns in the output table:

Port information

The **PPrt** column of the network device view identifies physical RoCE Express adapter ports with numerical identifiers, starting from 0. With two ports per adapter, the value is either 0 or 1.

The **IPrt** column of the InfiniBand view identifies ports by PCI function with numerical identifiers, starting from 1. Because each PCI function has exactly one port, the value is always 1.

Device information

The **Net-Dev** column of the network device view shows the interface name for the PCI function.

The **IB-Dev** column of the InfiniBand view shows the device name as used by the device driver that handles the PCI function. For PCI functions that have been prepared as PCI pass-through devices on a KVM host, `n/a` is shown instead of a device name.

Both views show the PNET ID, if assigned in the hardware configuration. PNET IDs represent a user-defined logical ID of a physical network. For PCI functions, PNET IDs are relevant to SMC-R connections (see "Using a PCI function for SMC-R connections" on page 19) and to HSCI interfaces (see "Using a PCI function for HSCI connections" on page 20).

For smc-tools earlier than v1.8, transient PNET IDs that are assigned with the **smc_pnet** command are not shown in the **smc_rnics** output. Issue **smc_pnet** without command arguments to display the transient PNET IDs that are assigned on your Linux instance.

You can find the UID of a PCI function from the function address that is displayed in the **PCI_ID** column of the **smc_rnics** command. Read the `uid` attribute for the PCI function in the PCI branch of sysfs. Issue a command of this form:

```
# cat /sys/bus/pci/devices/<function_address>/uid
```

In the following example, the UID of a PCI function with function address `0008:00:00.0` is `0x8`.

```
# cat /sys/bus/pci/devices/0008:00:00.0/uid
0x8
```

Network interface names

As of Linux kernel 5.13 and with the udev rules of systemd 249, distributions can use predictable and persistent interface names for your RoCE Express PCI functions.

Predictable interface names are preferably based on UIDs. Only if unique UIDs are not guaranteed, predictable interface names are based on FIDs. Kernels that support UID-based predictable interface names have a sysfs attribute `/sys/bus/pci/devices/`*`<function_address>`*`/uid_is_unique` for its PCI functions. In this path, *<function_address>* is the PCI ID that identifies a PCI function in the PCI stack.

The availability of predictable interface names on a particular distribution depends on whether the distribution supports and adopts these names. For UID-based predictable interface names, UID uniqueness must be enforced in your environment.

UID uniqueness

How UID uniqueness can be enforced depends on your hypervisor environment.

LPAR
: The hardware administrator must set the **UID uniqueness** option in the hardware configuration of the partition.

  For more information, see the section about defining partitions in *z/OS HCD User's Guide*, SC34-2669.

DPM partition
: UID uniqueness is always enforced.

KVM virtual server
: UID uniqueness is always enforced.

z/VM guest virtual machine
: UID uniqueness must be enforced in both the partition and in z/VM.

  In z/VM, you control UID uniqueness with the `ENFORCE_BY_VOLID` option of the **`SET IO_OPT`** CP command. By default, UID uniqueness is enforced. For more information, see *z/VM: CP Commands and Utilities Reference*, SC24-6268.

  For z/VM in LPAR mode, ensure that UID uniqueness is enforced in the hardware configuration of the LPAR.

  Enabled UID uniqueness for z/VM in an LPAR that does not enforce it results in misleading values of the `uid_is_unique` attribute and to limitations for your PCI functions.

To confirm that your Linux instance runs in a partition for which UID uniqueness is enforced, read the `uid_is_unique` attribute of any PCI device in sysfs. The following example returns the value 1 for a PCI device 0001:00:00.0, which means that UID uniqueness is enforced.

```
# cat /sys/bus/pci/devices/0001:00:00.0/uid_is_unique
1
```

z/VM: For z/VM guests, this value is can be misleading unless z/VM and the LPAR have matching settings for enforcing or not enforcing UID uniqueness.

If it is supported, ensure that UID uniqueness is enforced to attain the most convenient interface names for your environment.

Predictable names

Predictable interface names use one of two base names, depending on whether UID uniqueness is enforced on the Linux instance.

eno
: With enforced UID uniqueness, interface names depend only on the UID. The format is eno*`<UID_in_dec>`*, where *<UID_in_dec>* is the UID in decimal notation.

This is the preferred naming scheme. Enforce UID uniqueness if possible.

ens
Without enforced UID uniqueness, interface names depend only on the FID. The format is `ens`*`<FID_in_dec>`*, where *<FID_in_dec>* is the FID in decimal notation.

The following table shows some examples for UIDs and FIDs and the resulting network interface names.

UID and FID of the PCI function	With enforced UID uniqueness (preferred)	Without enforced UID uniqueness
UID: 0x0, FID: 0x0	eno0	ens0
UID: 0x5, FID: 0x7	eno5	ens7
UID: 0xa, FID: 0xb	eno10	ens11
UID: 0x10, FID: 0x20	eno16	ens32

The eth*<n>* naming scheme

Distributions that do not adopt predictable interface names might use the earlier eth*<n>* scheme.

Such distributions typically assure a persistent mapping of PCI functions to interface names through distribution-specific tooling. On other distributions, forcing the eth*<n>* naming scheme with the `net.ifnames=0` can lead to a nonpersistent mapping.

Other naming schemes

Distributions for which the predictable interface naming scheme does not take effect might use one of the following naming schemes:

enP*<UID_or_counter>*s*<FID_in_dec>*
enP*<UID_or_counter>*p0s
enP*<UID_or_counter>*p0np0
enP*<UID_or_counter>*p0s0np0
enP*<UID_or_counter>*p0

This list is not exhaustive, and even the `ens`*`<FID_in_dec>`* pattern might be used.

In these schemes, *<UID_or_counter>* represents the UID, in decimal notation, in environments where UID uniqueness is enforced. Without UID uniqueness, a hexadecimal counter is used. *<FID_in_dec>* represents the FID in decimal notation.

For example, the interface name for a PCI function with FID `0x8ca` and UID `0x10` could be `enP16s2250`, `enP0s2250`, `enP1s2250`, ... `enP7s2250`, `enP16p0s`, `enP0p0s`, `enP1p0s`, ... `enP16p0np0` ...

The multiple dependencies on the hardware, on the hardware configuration, on the Linux kernel, and on the udev-rules of the distribution makes it difficult and error prone to predict interface names that follow these schemes.

altnames

Distributions might include interface names for multiple naming schemes as altnames. Such altnames can include predictable interface names. Always use the primary interface name for your network configuration. For example, you might not be able to use altnames for persistent configurations.

The output of **`ip a`** shows altnames, if present, for your network interfaces.

```
# ip a
1: lo: ...
   ...
2: eno4272: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
   link/ether 82:11:a1:b9:82:06 brd ff:ff:ff:ff:ff:ff
   altname ens993
   altname enP4272p0s0
   ...
```

Online and configuration state

Changing the online state of a PCI function within Linux in LPAR or DPM partition mode also changes the function's configuration state in the partition.

In contrast, changing the online state of a PCI function in KVM or z/VM guests has no effect on the function's configuration state in the partition. KVM or z/VM guests depend on the hypervisor to control the configuration state. For more information about controlling the configuration state of PCI functions from z/VM, see the z/VM documentation.

PCI functions cannot be used concurrently by operating systems in multiple partitions. If the hardware configuration makes a PCI function eligible for multiple partitions, the function becomes exclusively assigned to the first partition that configures it. This assignment persists, across POR cycles of the partition, until the PCI function is explicitly deconfigured. Setting a PCI function offline from Linux in LPAR mode also deconfigures it in the partition and so makes it available to other eligible partitions.

During the installation of a new Linux instance, the installer requires a network connection. If RoCE Express is the only available network adapter, one of its PCI functions must be configured before the installation begins, see "Using a PCI function during a Linux installation" on page 16.

Chapter 2. Managing PCI functions on Linux on IBM Z

Make PCI functions of RoCE Express available to your Linux instance and manage them from Linux.

Hardware environment and settings

Hardware defaults and settings govern the following conditions for RoCE Express PCI functions in your LPAR or DPM partition:

- Which PCI functions are eligible for your partition.
- Which of these PCI functions are also eligible for other partitions.
- Whether unique UIDs for each PCI function are enforced in your partition.
- The initial configuration state of PCI functions in your partition.

Partitioning hardware resources is beyond the scope of this publication. For more information, see, for example, *z/OS HCD User's Guide*, SC34-2669.

Making PCI functions available to Linux

Hardware settings define which PCI functions are available to Linux instances that run in LPAR or DPM partition mode. For Linux as a KVM or z/VM guest, use hypervisor techniques to pass PCI functions on to the guest.

Configuring a PCI function for a KVM virtual server

Configure the pass-through device for the KVM virtual server and define the configuration to libvirt.

Before you begin

The PCI function must be configured in the partition in which the KVM host runs, see "Online and configuration state" on page 10.

About this task

You can use a PCI function to back one or more generic virtio network devices on one or more virtual servers through MacVTap interfaces. Because you cannot run the network interfaces of RoCE Express PCI functions in promiscuous mode, Open vSwitch or a Linux bridge are not an option.

You can directly use the network interface of the PCI function or use a bonded interface on the host. For more information about MacVTap connections, see *KVM Virtual Server Management*, SC34-2752.

The steps that follow describe how to use a PCI function as the host resource for a VFIO pass-through device. For PCI functions as VFIO pass-through devices, you need QEMU v2.11 and libvirt v4.10 on the KVM host.

Procedure

1. Find out the function address of the PCI function.

   Example: The commands of the following example display this information for a PCI function with FID `0x050a`.

```
# smc_rnics -a
  FID Power PCI_ID       PCHID Type          PPrt PNET_ID Net-Dev
-------------------------------------------------------------------
...
  50a 1     000a:00:00.0 015c  RoCE_Express2 0    NET20   eno10
...
```

If the `smc_rnics -a` output line for FID `50a` shows a value `0` in the power column as the only information, issue `smc_rnics -e 50a` to configure the PCI function for the partition and set it online in Linux.

2. Specify a device configuration for the PCI pass-through device, within the domain configuration-XML of the virtual server or as a separate device configuration-XML.

   Tip: In the specification, use the `managed="yes"` attribute to simplify the host setup for your PCI function. Without this specification, you must free the PCI function from control of its default device driver on the KVM host. You then must enable the vfio_pci device driver to handle your PCI function.

   Example: This example illustrates how the PCI function `000a:00:00.0` of the host is encoded in the address element of the source specification.

```
<hostdev mode="subsystem" type="pci" managed="yes">
   <source>
      <address domain="0x000a" bus="0x00" slot="0x00" function="0x0"/>
   </source>
</hostdev>
```

3. Define the virtual server or separately defined device to libvirt with the **`virsh define`** command.

   When libvirt processes the device specifications, it adds specifications that identify the PCI function to the KVM guest. These specifications include a UID and an FID.

   Example: After defining a device with the specifications to libvirt, the example of step "2" on page 12 might be expanded like this:

```
<hostdev mode="subsystem" type="pci" managed="yes">
   <source>
      <address domain="0x000a" bus="0x00" slot="0x00" function="0x0"/>
   </source>
   <address type="pci" domain="0x0001" bus="0x00" slot="0x00" function="0x0">
      <zpci uid="0x0001" fid="0x00000000"/>
   </address>
</hostdev>
```

   Accept the attribute values in the newly added address element. You can change the UID and FID in the nested zpci element according to your needs.

4. For a separately defined device, attach the device to the virtual server with the **`virsh attach-device`** command.

Results

The PCI function is now available and online in your KVM guest. For more details and options for managing PCI pass-through devices, see *KVM Virtual Server Management*, SC34-2752.

Attaching a PCI function to a z/VM guest virtual machine

Use the z/VM CP ATTACH command to make PCI functions available to z/VM guests.

Before you begin

The PCI function must be configured in the partition in which the z/VM hypervisor runs, see "Online and configuration state" on page 10.

The z/VM system must be configured to support PCI functions. For more information, see the section about PCIe functions in *z/VM: CP Planning and Administration*, SC24-6271.

Live-guest migration of a z/VM guest within a z/VM single system image (SSI) is not possible if the z/VM virtual machine has a directly attached PCI function.

About this task

The steps that follow describe z/VM CP commands as issued from a CMS session in a z/VM guest virtual machine. Using the **vmcp** command, you can issue these CP commands from a Linux instance that runs as a guest in that z/VM guest virtual machine.

Procedure

1. Optional: Confirm that the PCI function to be attached to the z/VM guest is available to your z/VM system.

   From a z/VM user ID with administrator privileges, issue:

   ```
   #CP QUERY PCIFUNCTION
   ```

   The output lines begin with PCIF followed by the FID of the PCI function in hexadecimal notation.
2. Attach the PCI function by issuing a CP ATTACH command of this form:

   ```
   #CP ATTACH PCIFUNCTION <fid> TO <zvm_userid>
   ```

   Where:

   <fid>
   : is the function ID of the PCI function as used in the hardware configuration and as shown by the CP QUERY command.

   <zvm_userid>
   : is the z/VM user ID of the z/VM guest virtual machine.

   You can map the FID in the partition to an FID of your choice in the z/VM guest virtual machine. To map to a different FID, append the CP command with `TO` *<vm_fid>* to use *<vm_fid>* within the guest.

   Example:

   ```
   #CP ATTACH PCIFUNCTION 8CA TO P3VM5LZ1
   ```

What to do next

For more information about managing PCI functions on z/VM, see *z/VM: CP Commands and Utilities Reference*, SC24-6268.

Listing PCI functions

Use the **`lspci`** or **`smc_rnics`** command to list your PCI functions.

Listing online PCI functions with `lspci`

The **`lspci`** command lists all online PCI functions on a running Linux instance, including PCI functions of RoCE Express adapters.

```
# lspci |grep Ethernet
```

For more information about the **`lspci`** command, see the man page.

Listing all PCI functions with `smc_rnics`

By default, the **`smc_rnics`** command lists all online PCI functions of RoCE Express adapters.

The command has a `--all` option that includes offline PCI functions in the output list. Because the command cannot detect the type of offline PCI functions the output cannot be filtered to items that are relevant to SMC. The listed offline device could include, for example, NVMe devices. Such devices are omitted from the **`smc_rnics`** output after being set online.

Example

This example assumes four PCI functions with the following FIDs:

50a
: is an offline RoCE Express PCI function with function address `000a:00:00.0`.

8ca
: is an online RoCE Express PCI function with function address `0008:00:00.0`.

8ea
: is an online RoCE Express PCI function with function address `0009:00:00.0`.

b05
: is an offline NVMe device with function address `000d:00:00.0`.

By default, both the **`lspci`** command and the **`smc_rnics`** command list the two online PCI functions. With the `--all` option, **`smc_rnics`** also lists the offline FIDs.

```
# lspci
0008:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4 Lx Virtual
Function]
0009:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4 Lx Virtual
Function]
# smc_rnics
  FID Power PCI_ID       PCHID Type          PPrt PNET_ID Net-Dev
-------------------------------------------------------------------
  8ca 1     0008:00:00.0 01c8  RoCE_Express2 0    NET25   eno8
  8ea 1     0009:00:00.0 01c8  RoCE_Express2 1    NET26   eno9
# smc_rnics --all
  FID Power PCI_ID       PCHID Type          PPrt PNET_ID Net-Dev
-------------------------------------------------------------------
  50a 0
  8ca 1     0008:00:00.0 01c8  RoCE_Express2 0    NET25   eno8
  8ea 1     0009:00:00.0 01c8  RoCE_Express2 1    NET26   eno9
  b05 0
```

The following command outputs assume that FIDs 50a and b05 are set online, for example, with **`smc_rnics -e`** commands. The **`smc_rnics`** command now filters out the NVMe device, but **`lspci`** lists it.

```
# smc_rnics
  FID Power PCI_ID       PCHID Type          PPrt PNET_ID Net-Dev
-------------------------------------------------------------------
  50a 1     000a:00:00.0 01a5  RoCE_Express2 1    NET26   eno10
  8ca 1     0008:00:00.0 01c8  RoCE_Express2 0    NET25   eno8
  8ea 1     0009:00:00.0 01c8  RoCE_Express2 1    NET26   eno9
# lspci
0008:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4 Lx Virtual
Function]
0009:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4 Lx Virtual
Function]
000a:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4 Lx Virtual
Function]
000d:00:00.0 Non-Volatile memory controller: Samsung Electronics Co Ltd NVMe SSD Controller 172Xa/
172Xb (rev 01)
```

For KVM or z/VM guests, PCI devices are online by default.

Controlling the online state in Linux

Use the **`smc_rnics`** command or the sysfs power attribute of a PCI function to set it online or offline in Linux.

Before you begin

Changing the online state of a PCI function from Linux in LPAR or DPM partition mode affects the configuration state of the PCI function in the partition, see "Online and configuration state" on page 10.

Procedure

1. Optional: List your PCI functions with the **smc_rnics** command to assure that the PCI function of interest is available to your partition. In the command, specify the -a option to include offline PCI functions in the command output.

   The command displays a table of all PCI functions that are eligible for your partition. A 0 in the Power column of the output table indicates that the PCI function is offline.

   Example: This example shows an offline PCI function with FID 0x050a.

```
# smc_rnics -a
  FID Power PCI_ID       PCHID Type          PPrt PNET_ID Net-Dev
-------------------------------------------------------------------
...
  50a 0
...
```

2. Set the PCI function online by issuing an **smc_rnics** command with the -e option or through sysfs.

   - Issue a command of the following form:

     ```
     # smc_rnics -e <fid>
     ```

     where *<fid>* specifies the FID in hexadecimal format. You can omit leading zeros.

     Hint: If the PCI function is already configured in another partition, you cannot set it online and the command fails.

     Example: To set a PCI function with FID 0x050a online, issue:

     ```
     # smc_rnics -e 50a
     ```

   - Alternatively, write 1 to the sysfs /sys/bus/pci/slots/*<fid>*/power attribute of the PCI function. In this path, *<fid>* is the function's FID as an 8-digit hexadecimal number.

     Example: To set a PCI function with FID 0x050a online, issue:

     ```
     # echo 1 > /sys/bus/pci/slots/0000050a/power
     ```

   The PCI function is now online to your Linux instance. The output table of the **smc_rnics** command now shows more information about the PCI function and 1 is displayed in the Power column.

   Example:

```
# smc_rnics
  FID Power PCI_ID       PCHID Type          PPrt PNET_ID Net-Dev
-------------------------------------------------------------------
...
  50a 1     000a:00:00.0 015c  RoCE_Express  0    NET20   eno10
...
```

What to do next

You can now use the **ip** command to activate the interface that is provided by the PCI function. You can see the interface name in the **Net-Dev** column of your **smc_rnics** output table, eno10 in the example.

You can use the **ethtool** and **ip** commands for a transient configuration of your network interface. For a persistent configuration, use the network configuration tools of your distribution. For information about these tools, see your distributor's documentation.

To set a PCI function offline, use the **smc_rnics** command with the -d. Alternatively, write 0 to the sysfs power attribute of the PCI function.

Using a PCI function during a Linux installation

An operational network device is a prerequisite for installing Linux on IBM Z or LinuxONE systems. You can use a RoCE Express PCI function to provide this device.

Assuring that the PCI function is available

For installing Linux as a KVM or z/VM guest, you must assure that the hypervisor makes the PCI function available to the virtual server or guest virtual machine. No further action is needed.

For installations in an LPAR or DPM partition, the PCI functions must be configured for the partition, which depends on several conditions.

- The hardware system defaults or settings might configure the PCI function for the partition. For example, if the PCI function is listed in an LPAR access list, it is automatically configured when the LPAR is started.
- For DPM partitions, you can specify the PCI function for auto-configuration when an operating system is started. For more information, see the section about device auto-configuration in *Device Drivers, Features, and Commands*.
- If the PCI function was online in the operating system that previously ran in the partition, it remains configured for the partition.

If the PCI function is not configured for the partition, use hardware management interfaces to configure it, see "Using the SE or HMC to configure a PCI function for a partition" on page 17.

Specifying the network interface name for the installer

Depending on your distribution, you can specify the network interface during the installation. Your distribution might display the available network interfaces on the Operating System Messages applet of the SE or HMC and prompt you to select one. For installations of KVM or z/VM guests, you might be prompted in the KVM host or z/VM CP session that drives the installation.

For other distributions, you must specify suitable kernel parameters for the installer. For example, for a Red Hat Enterprise Linux installation, you can specify the interface name with the `ip=` or `vlan=` kernel parameter. If the network conventions and regulations at your installation permit it, you can use DHCP to get an automatic IP configuration.

Network interface names for RoCE Express PCI functions depend on your environment, see "Network interface names" on page 8. If your Linux instance uses predictable interface names, specify the expected name. For example, specify eno8 if UID uniqueness is enforced and the UID of your PCI function is 0x8.

Tip: If you are not sure about the network interface name, specify your best guess and observe the kernel messages that are displayed on the SE or HMC during the installation. Watch out for a message like this and, if necessary, repeat the installation with the correct name.

```
...
[...] mlx5_core 0008:00:00.0: enP8p0s0: renamed from eth0
...
```

Hint: If there is no such line, confirm that the PCI function is available to the installer. Assure that the PCI function configured for your partition or that the KVM or z/VM hypervisor makes it available to the virtual server or guest virtual machine.

Installer specification example for Red Hat Enterprise Linux 8

In a `.prm` file, add a line like this to specify eno8 as the interface name:

```
ip=10.10.92.31::10.10.92.30:16:<lpar>:eno8:none
```

Setting the MTU size

A RoCE Express PCI function has two separate maximum transmission units (MTUs).

- MTU for the networking interface.
- MTU for the InfiniBand interface.

You explicitly set the MTU for the networking interface. Use the **ip** command for a transient configuration or distribution tools for a persistent configuration. The following example sets the MTU of the networking interface eno10 to 1500 bytes.

```
# ip link set dev eno10 mtu 1500
```

You cannot explicitly set the MTU for the InfiniBand interface. This MTU is derived from the MTU of the networking interface. The possible values depend on the adapter version. For example, RoCE Express2 adapters support 1024 bytes, 2048 bytes, and 4096 bytes. Of these possible values, the largest that does not exceed the MTU of the networking interface is used.

For example, for a PCI function of RoCE Express2 and an MTU of 1500 bytes for the networking interface, the MTU for the InfiniBand interface is 1024 bytes.

Use the **ibv_devinfo** command to display the MTU of the InfiniBand interface as shown in the following example.

```
# ibv_devinfo
hca_id: mlx5_0
       transport:                      InfiniBand (0)
       ...
       phys_port_cnt:                  1
               port:   1
                       state:                  PORT_ACTIVE (4)
                       max_mtu:                4096 (5)
                       active_mtu:             1024 (3)
                       ...
```

In the command output, the MTU information has the following meaning:

active_mtu
is the MTU that is used for the InfiniBand interface.

max_mtu
is the maximum MTU that is supported by the adapter hardware.

The MTU of the networking interface is not shown in the command output.

Using the SE or HMC to configure a PCI function for a partition

You can use the IBM Z or LinuxONE Support Element (SE) or an attached Hardware Management Console (HMC) to control the configuration state of PCI functions in an LPAR or DPM partition.

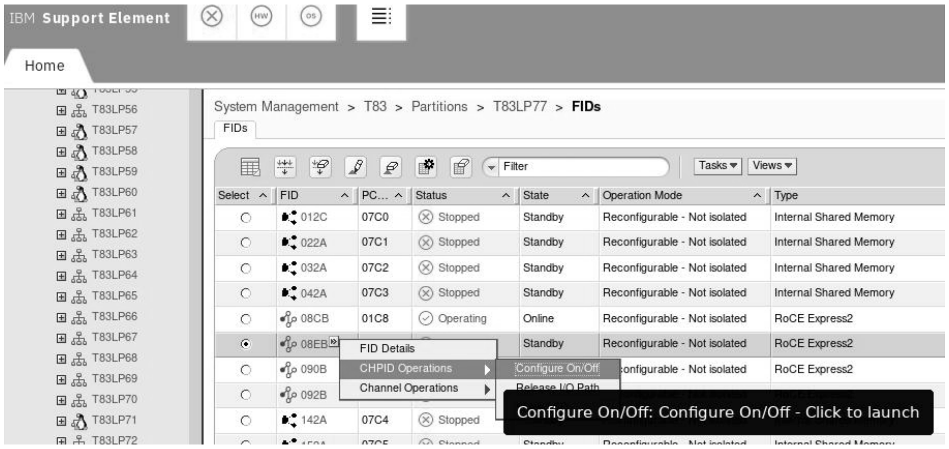
Before you begin

You can control the configuration state of PCI functions in a partition from a Linux instance that runs in the partition in LPAR or DPM partition mode, see "Controlling the online state in Linux" on page 14.

The steps that follow are primarily intended to configure a PCI function for a partition, as a prerequisite for a Linux installation (see "Using a PCI function during a Linux installation" on page 16). The steps are based on an SE interface but after selecting the hardware system, are similar on an HMC.

Procedure

1. If you work from an HMC, select your partition and select **recovery** > **single object operations**.
2. Open the **FIDs** tab.



System Management > T83 > Partitions > T83LP77 > **FIDs**

Select	FID	PC...	Status	State	Operation Mode	Type
○	012C	07C0	Stopped	Standby	Reconfigurable - Not isolated	Internal Shared Memory
○	022A	07C1	Stopped	Standby	Reconfigurable - Not isolated	Internal Shared Memory
○	032A	07C2	Stopped	Standby	Reconfigurable - Not isolated	Internal Shared Memory
○	042A	07C3	Stopped	Standby	Reconfigurable - Not isolated	Internal Shared Memory
○	08CB	01C8	Operating	Online	Reconfigurable - Not isolated	RoCE Express2
●	08EB			Standby	Reconfigurable - Not isolated	RoCE Express2
○	090B				Reconfigurable - Not isolated	RoCE Express2
○	092B					
○	142A	07C4	Stopped			

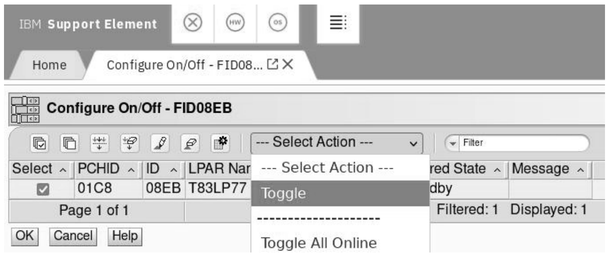
Configure On/Off: Configure On/Off - Click to launch

Figure 3. FIDs tab on the SE

As shown in Figure 3 on page 18, PCI functions are identified through FIDs. The value in the **Type** column confirms that an FID corresponds to a PCI function of a RoCE Express adapter.

Tip: You can select multiple FIDs to proceed with configuring multiple PCI functions for the partition.

3. Select your FID, then click **Tasks** > **CHPID Operations** > **Configure On/Off** to open the **Configure On/Off** tab.



Configure On/Off - FID08EB

Select	PCHID	ID	LPAR Nar	red State	Message
✓	01C8	08EB	T83LP77	dby	

Page 1 of 1 Filtered: 1 Displayed: 1

Figure 4. FIDs tab on the SE

4. Select the FID or FIDs that you want to configure, then click **--- Select Action ---** > **Toggle**.

   The state in the **Desired State** column changes from `Standby` to `Online`.

5. Click **OK**.

   The state in the **Current State** column changes from `Standby` to `Online`.

   This operation fails if the PCI function is already online in another partition.

Displaying the RDMA information for a PCI function

Use the **`ibv_devinfo`** command to display RDMA information for your PCI function. Depending on your distribution, the package that includes this command might be called `libibverbs-utils` or `ibverbs-utils`.

About this task

RDMA is based on the InfiniBand communications standard, so the RDMA properties of a RoCE Express PCI function include InfiniBand properties.

Procedure

1. Issue **`ibv_devinfo -l`** to list your PCI functions.

   Example: This example lists 4 PCI functions:

```
# ibv_devinfo -l
4 HCAs found:
        mlx5_0
        mlx5_1
        mlx5_2
        mlx5_3
```

The listed identifiers match the InfiniBand device names as listed by the **smc_rnics** command with the `-I` option.

2. Issue **ibv_devinfo -d**, followed by the PCI function of interest as listed by **ibv_devinfo -l**.

   Example: This example list details for the PCI function with device name `mlx5_1`.

```
# ibv_devinfo -d mlx5_1
hca_id: mlx5_1
        transport:                  InfiniBand (0)
        fw_ver:                     14.25.1020
        node_guid:                  9928:1bfe:ff9b:1082
        sys_image_guid:             9803:9b03:001b:2898
        vendor_id:                  0x02c9
        vendor_part_id:             4118
        hw_ver:                     0x0
        board_id:                   IBM0000000016
        phys_port_cnt:              1
                port:                       1
                state:                      PORT_ACTIVE (4)
                max_mtu:                    4096 (5)
                active_mtu:                 1024 (3)
                sm_lid:                     0
                port_lid:                   0
                port_lmc: 0x00
                link_layer: Ethernet
```

The displayed information includes the maximum supported MTU and the current, active MTU (see also "Setting the MTU size" on page 17). The state attribute indicates whether an active connection is using the port.

Using a PCI function for SMC-R connections

Start your application with the **smc_run** command to enable it for SMC-R connections.

The **smc_run** command uses the `libsmc-preload.so` preload library for the application. This library makes existing TCP/IP socket programs use SMC if both communication peers support it. After an initial handshaking through a TCP/IP connection, the PCI function is used in RDMA mode for all further traffic.

Before you begin

Both the **smc_run** command and the `libsmc-preload.so` preload library are included in the smc-tools package.

About this task

To establish an SMC-R connection you need two interfaces: a TCP/IP interface and an RDMA interface. The two interfaces must be associated through a matching PNET ID. PNET IDs label interfaces as belonging to a particular physical network, see "Managing PNET IDs" on page 26.

Because RoCE Express PCI functions on Linux on IBM Z can provide both interfaces you have a choice.

- Use a TCP/IP interface from an OSA adapter and associate it with PCI function for the RDMA interface.
- Use the PCI function for both interfaces.

  For this choice, assigning the appropriate PNET ID merely labels the PCI function as belonging to a physical network. You do not need a PNET ID to associate the TCP/IP and RDMA interface of the same PCI function.

If a connection is opened while no functional SMC-R link is available, TCP/IP is used. For established connections, SMC-R links do not revert to TCP/IP.

Configure for link failover by assigning the same PNET ID to multiple PCI functions that can reach the intended peer. For such setups, the TCP/IP link must be based on an OSA adapter. Typical high availability setups use two bonded TCP/IP interfaces, each from a different OSA adapter, and two PCI functions from different resource groups, all associated with the same PNET ID (see "Path redundancy for SMC-R connections" on page 23).

Procedure

1. Confirm that your communication network, the network interfaces on your Linux instance, and the intended communication peer are all set up correctly for an SMC-R connection.

   Issue an **smc_chk** command to confirm the setup.

   Resolve any issues until the **smc_chk** command confirms the setup.

```
# smc_chk -C 192.168.5.47 -p 23
  Live test (SMC-D and SMC-R)
     Success, using SMC-R
```

2. Start your application with the **smc_run** command according to the following syntax.

```
# smc_run <smc_run_parameters> <program> <program_parameters>
```

   For more information about the **smc_run** command, see the man page or issue **smc_run -h**.

Example

This example specifies two **smc_run** parameters, `-r` to set a receive buffer to 16384 bytes, and `-t` to set a transfer buffer to 512 KB. The command starts a program iperf3 with program-specific parameters `-s` and `-p`.

```
# smc_run -r 16384 -t 512k iperf3 -s -p 12345
```

Using a PCI function for HSCI connections

A PCI function can provide the external interface for a HiperSockets Converged Interface (HSCI) connection.

HSCI interfaces integrate HiperSockets connectivity with your external LAN into a single logical network. For more information about HSCI, see *Device Drivers, Features, and Commands*.

Traffic through an HSCI interface automatically uses the best available connection to a communication peer with the same PNET ID: HiperSockets if the peer can be reached through HiperSockets, and the PCI function for other peers.

Before you begin

- HSCI is supported as of z15 and LinuxONE III.
- Your Linux instance must run in LPAR or DPM partition mode.
- Both the interface for the PCI function and for the HiperSockets device must be up, but no IP address must be assigned.
- Preferably, the hardware configuration assigns the same PNET ID to both the PCI function and the HiperSockets device, see "Managing PNET IDs" on page 26.

Procedure

1. Optional: Assure that your PCI function and your HiperSockets device do not have diverging PNET IDs.

PNET IDs are not required for HSCI, but if both the PCI function and the HiperSockets have PNET IDs, they must match.

For PCI functions issue the **smc_rnics** command or use **smc_chk**. For HiperSockets devices, use **smc_chk**.

The following example checks the PNET IDs of a PCI function with interface name eno5 and a HiperSockets device with interface name hsi3.

```
# smc_chk -i eno5
NET100
# smc_chk -i hsi3
NET100
```

2. Create an HSCI interface by issuing the **hsci add** command with the HiperSockets interface and the PCI function interface as arguments. Specify the HiperSockets interface first.

   The created HSCI interface has a name hsci*<devno>*, where *<devno>* is the device number of the HiperSockets device.

Example

The command in this example creates an HSCI interface from a HiperSockets interface hsi3 and a PCI function interface eno5. The HiperSockets device number is assumed to be 8410.

```
# hsci add hsi3 eno5
Verifying net dev eno5 and HiperSockets dev hsi3
Adding hsci8410 with a HiperSockets dev hsi3 and an external dev eno5
Added HSCI interface hsci8410
```

What to do next

You can now assign an IP address to the HSCI interface.

Troubleshooting for PCI functions

Try to reset defective PCI functions. You might also have to set PCI functions offline in preparation for upgrades or physical repair actions by the hardware administrator.

Use the **smc_rnics** command with the -d option to set a defective PCI function offline. Subsequently, use the **smc_rnics** command with the -e option to set it back online. For Linux in LPAR or DPM partition mode, these commands also deconfigure and configure the PCI function in the partition.

If a PCI function does not respond to the **smc_rnics** command, proceed depending on your environment.

Linux as a KVM guest

Set the PCI function offline in the guest, then use **smc_rnics** and, if needed, **zpcictl** on the host.

Linux in LPAR or DPM partition mode

Use the **zpcictl** command with the `--reset` option to force a reset.

If the reset fails, you can use the **zpcictl** command with the `--deconfigure` option to deconfigure the PCI function by force. This command performs a crude, unplug-style removal of the PCI function. Do not use it for operational PCI functions.

Linux as a z/VM guest

Issue z/VM CP commands to detach the PCI function from the z/VM virtual machine, deconfigure it, configure it afresh, and re-attach it.

The following example uses the **vmcp** command to issue the CP commands from Linux. The PCI function of the example has FID 2a3.

```
# vmcp 'DETACH PCIF 2A3 FORCE'
# vmcp 'CP VARY OFF PCIF 2A3'
# vmcp 'VARY ON PCIF 2A3'
# vmcp 'ATTACH PCIF 2A3 TO *'
```

If you cannot recover the defective PCI function report the problem to the hardware administrator.

Chapter 3. Best practices

Set up a performant and resilient networking topology with RoCE Express.

High availability setup

High availability setups for applications on Linux on IBM Z or LinuxONE typically use path redundancy for network connections.

Identifying suitable PCI functions

To avoid outages during resource group and RoCE Express adapter maintenance, use redundant paths through PCI functions that are managed by different resource groups. Such PCI functions have different values in `/sys/bus/pci/devices/<pci_id>/pfip/segment0`, where *<pci_id>* is the function address. PCI functions that are managed by different resource groups are always based on different RoCE Express adapters.

In the following example, the PCI functions with function addresses `000d:00:00.0` and `00b5:00:00.0` are on different resource groups:

```
# cat /sys/bus/pci/devices/000d:00:00.0/pfip/segment0
0x01
# cat /sys/bus/pci/devices/00b5:00:00.0/pfip/segment0
0x03
```

Path redundancy for TCP/IP connections

On Linux, combine network interfaces for redundant paths into a bonded, single network interface that can then be used by applications. As of kernel 4.14, you can use the bonding device driver to create bonded interfaces.

For more information about bonded interfaces, see *Linux Channel Bonding Best Practices and Recommendations*. You can find this publication on IBM Documentation at www.ibm.com/docs/en/linux-on-systems?topic=availability-linux-channel-bonding-best-practices-recommendations. This publication describes bonding of OSA-Express based interfaces, but the descriptions of the bonding device driver also apply to RoCE Express based interfaces. In particular, as for OSA-Express, the BONDING_MODULE_OPTS specification must include `fail_over_mac` option. The exact option name can vary by distribution.

For KVM guests, you can use a bonded interface in the KVM host for a MacVTap connection. For more information about MacVTap connections, see *KVM Virtual Server Management*, SC34-2752.

KVM or z/VM guests cannot attain path redundancy through virtual switches in the hypervisor.

z/VM
VSWITCH does not support RoCE Express.

KVM
RoCE Express does not provide a promiscuous mode as required for Open vSwitch.

Path redundancy for SMC-R connections

Use SMC-R link groups to guard against link failure in SMC-R connections. SMC-R automatically creates link groups for PCI functions with matching PNET IDs.

To safeguard against failure of the TCP/IP connection, use a bonded interface that combines paths through two different OSA-Express adapters.

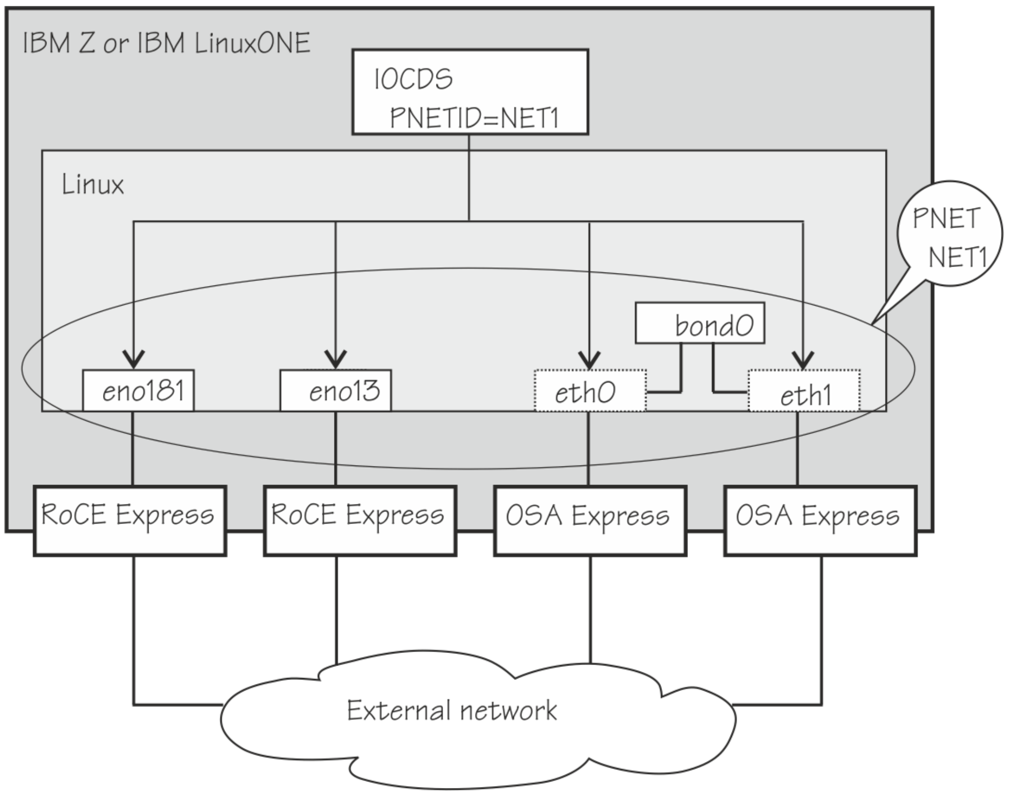


Figure 5. HA setup for SMC-R

Figure 5 on page 24 shows a Linux instance with four paths to an external LAN.

Two paths use PCI functions from separate RoCE Express adapters with different resource groups. In Linux, these paths result in network interfaces eno181 and eno13.

The paths through the two OSA Express adapters result in network interfaces `eth0` and `eth1`. These two interfaces are bonded into an interface `bond0`.

In its IOCDS, the hardware configuration assigns the same PNET ID, PNET1, to eno181, eno13, `eth0`, and `eth1`. This common PNET ID associates the four interfaces, and by extension also the bonded interface `bond0`. The two RoCE Express interfaces form an SMC link group.

A connection that is initiated through `bond0` has a redundant TCP/IP connection. The SMC link group provides failover for RDMA traffic.

Performance tuning

Performance optimizations are strongly dependent on your workload and on your hardware and software environment.
The suggestions that follow might or might not be suitable for your setup. Verify any settings against a suitable benchmark to avoid adverse tuning effects.

Tip: Use **`ethtool`** to verify that intended settings, like hardware offloads, are active. Also use **`ethtool`** to explore possible optimizations with transient settings before making them persistent through distribution tools.

Hardware offloads

By default, hardware offloads are enabled for checksums of both inbound and outbound packets and for TCP segmentation (TSO).

Use the **`ethtool`** command with the `-k` option to verify that the offloads are enabled.

Example:

```
# ethtool -k eno0
Features for eno0:
rx-checksumming: on
tx-checksumming: on
        tx-checksum-ipv4: off [fixed]
        tx-checksum-ip-generic: on
        tx-checksum-ipv6: off [fixed]
        tx-checksum-fcoe-crc: off [fixed]
        tx-checksum-sctp: off [fixed]
scatter-gather: on
        tx-scatter-gather: on
        tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: on
        tx-tcp-segmentation: on
...
```

In the command output, `[fixed]` labels settings that you cannot change.

Receive Packet Steering

For Linux instances with multiple CPUs, you can use Receive Packet Steering (RPS) to distribute incoming packets more evenly across specific CPUs.

By using hot caches, RPS can increase performance, especially with workloads that open numerous connections and transfer small packets.

This setting applies to network interfaces of all directly attached PCI functions, including PCI functions as VFIO pass-through devices on KVM guests.

For more information about RPS with RoCE Express, see *Exploring the performance of network adapters for Linux on IBM Z* [PDF].

Receive Flow Steering

Receive Flow Steering (RFS) is an extension of RPS. For selecting CPUs, RFS takes into account where the application runs that consumes an inbound packet.

For more information about RFS, see https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/network-rfs.

Enable Striding RQ

Help to use inbound buffers more efficiently, for example for inbound streaming workloads.

Example:

```
# ethtool --set-priv-flags eno0 rx_striding_rq on
# ethtool --show-priv-flags eno0
Private flags for eno0:
...
rx_striding_rq : on
...
```

Interrupt Moderation

Interrupt Moderation controls the waiting behavior on inbound packets before sending an interrupt. The settings manage the tradeoff between CPU cycles against latency and throughput. For most workloads, the default is a good compromise.

You can modify the waiting time with the `rx-usecs` and `tx-usecs` settings. You can modify the buffer count with the `rx-frames` and `tx-frames` settings.

Example:

```
# ethtool --show-priv-flags eno0
Coalesce parameters for eno0:
Adaptive RX: on TX: on
stats-block-usecs: 0
sample-interval: 0
pkt-rate-low: 0
pkt-rate-high: 0
rx-usecs: 8
rx-frames: 128
...
# ethtool -C eno0 rx-usecs 4095 rx-frames 65535 tx-usecs 4095 tx-frames 65535
```

Larger values mean saving CPU cycles by waiting longer and potentially collecting more inbound packets with each interrupt. Larger values also mean more latency. The default, "Adaptive RX" is optimized for latency. Only increase the waiting times and buffer counts if you want to save CPU cycles at the expense of latency.

For more information about interrupt moderation, see https://support.mellanox.com/s/article/understanding-interrupt-moderation.

Managing PNET IDs

PNET IDs represent a user-defined logical ID (name) of a physical network. The physical network represents the physical broadcast domain of a given network.

PNET IDs as hardware labels

Hardware administrators use PNET IDs within the hardware configuration to uniquely identify a physical layer 2 LAN fabric or physical broadcast domain. PNET IDs help to keep track of physical networks and the associated features, adapters, and ports.

PNET IDs must reflect networks that are in place through cabling and other network components. PNET IDs that are assigned to adapter ports, HiperSockets devices, and internal shared memory (ISM) devices can then be used by Linux to optimize network traffic through these components.

On Linux, you can use the **`smc_chk -i`** command to display the PNET ID for a specific interface, or the **`smc_rnics`** command to display all your PCI functions with any PNET IDs that are assigned in the hardware configuration.

PNET IDs for associating network interfaces

Within a Linux instance, matching PNET IDs associate network interfaces.

SMC-R link groups
For PCI functions with matching PNET IDs, SMC-R automatically creates an SMC-R link group. The link group provides link failover for the RDMA traffic of SMC-R connections.

OSA interfaces
If the PNET ID of an OSA interface matches that of an SMC-R link group or individual PCI function, the OSA interface can be used as the TCP/IP connection that is needed to initialize an SMC-R connection. Alternatively, you can use the PCI functions TCP/IP support for this initialization. On Linux, an OSA interface is an option for initializing an SMC-R connection, but it is not required.

Because the same PCI function can simultaneously provide TCP/IP and SMC-R connections, a PCI function can initiate SMC-R connections without a PNET ID. An appropriate PNET ID is still useful to label the PCI function as part of a physical network.

SMC-D devices
If the PNET ID of an ISM device matches that of an SMC-R link group or individual PCI function, SMC-D becomes an option for connecting peers with the same PNET ID. SMC-D then is the preferred option, followed by SMC-R and TCP/IP. For more information about SMC-D, see *Device Drivers, Features, and Commands*.

HiperSockets Converged Interface (HSCI)

No PNET ID is required to create an HSCI interface. However, you cannot create an HSCI that combines a PCI function and a HiperSockets device with different PNET IDs, see "Using a PCI function for HSCI connections" on page 20.

PNET IDs on Linux

Linux uses the PNET IDs of the hardware configuration to support HSCI interfaces or create SMC-R link groups. If the hardware configuration does not include all PNET IDs that you need, you can supplement them in a software PNET table.

Use the **smc_pnet** command to assign a PNET IDs through a PNET table. For example, to assign a PNET ID NET01 to a PCI function with network interface eno0, issue this command:

```
# smc_pnet -a NET01 --interface eno0
```

Follow these guidelines for assigning PNET IDs:

- Assure that the hardware does not already assign a PNET ID to the interface. If a PNET IDs is assigned in both the hardware configuration and in a PNET table, the PNET ID of the hardware configuration is used. Depending on your version of **smc_rnics**, you might get an error message if you try to assign a PNET ID to a PCI function for which one exists in the hardware configuration.

  Use the **smc_rnics** command to display your PCI functions with any PNET IDs that are assigned in the hardware configuration. As of smc-tools v1.8, the output also includes PNET IDs that are assigned through a PNET table.

  The **smc_rnics** output does not include PNET IDs of bonded interfaces. Bonded interfaces inherit the PNET IDs of their constituent interfaces if they have matching PNET IDs.
- To avoid confusion, align your PNET IDs with the labeling scheme that is used in the hardware configuration.
- Software PNET tables are intended to assign PNET IDs to entities that do not have a representation in the hardware configuration, for example to a MacVTap interface.

  You can also use PNET tables for testing connections, or when updating the hardware configuration would be too disruptive.

  PNET tables do not persist across reboots. Where possible, ensure that permanently required PNET IDs are assigned though the hardware configuration at the next opportunity.

Chapter 4. Migrating your network setup

You might need to migrate your network setup to use RoCE Express or to adapt an existing RoCE Express setup to a changed environment.

Attention: Migrating a network setup can be risky, especially if you are migrating the setup for the only connection to a Linux instance. You might not be able to use an inadequately migrated interface to connect to your Linux instance for corrective actions.

To mitigate against potential lockouts, set up and test an alternative connection to your Linux instance before you begin the migration. For example, use HiperSockets and a separate IP subnet for an alternative connection without any physical components. Once set up, verify that your alternative connection persists across reboots before you proceed to migrate your network setup.

You can resort to the Linux console to restore regular connectivity.

Linux as a KVM guest
Access the virtual console from the host or use virt-manager.

Linux as a z/VM guest
Access the console through a 3270 terminal emulation.

Linux in LPAR or DPM partition mode
Access the console through the **Operating System Messages** applet or through the **Integrated ASCII Console** applet on the HMC.

Migration to RoCE Express

Migrate from QDIO-based network interfaces to interfaces that are provided by PCI functions.

Before you begin

Your migration options depend on your current environment and how your OSA-based network interfaces are accessed by your Linux instance.

Directly attached OSA-Express adapters
Migration is possible for Linux in LPAR or DPM partition mode or for Linux as a z/VM guest.

KVM does not support directly attached OSA-Express adapters, but you can use RoCE Express PCI functions for VFIO pass-through devices.

OSA-Express-based interfaces connected through a z/VM VSWITCH
z/VM VSWITCH does not support RoCE Express. Continue to use QDIO-based interfaces with z/VM VSWITCH setups.

OSA-Express-based interfaces connected to Linux as a KVM guest through Open vSwitch
PCI functions do not have a promiscuous mode, so migration is not an option.

Goal

Seamlessly migrate the network traffic of a Linux instance from OSA-Express to RoCE Express.

Assumptions

NetworkManager was used to set up bonded interface, `mybond0`, on an instance of Red Hat Enterprise Linux. The bonded interface provides path redundancy through two OSA-Express-based network interfaces, `encbd00` and `encbd10`.

Two RoCE Express adapters are in place and PCI functions that are suitable to provide path redundancy are configured and visible in Linux. The corresponding network interfaces are `eno5968` and `eno6032`.

Strategy

First, add the RoCE Express based interfaces to the bonded interface. Then, remove the OSA-based interfaces.

Sample procedure

1. List your bonded interfaces.

```
# nmcli c
NAME                  UUID                                  TYPE      DEVICE
bond-mybond0          1f543303-4594-4230-9d2f-45a5eb424305  bond      mybond0
bond-slave-encbd00    30127166-31d8-4004-a55f-981df4d09573  ethernet  encbd00
bond-slave-encbd10    6ae85d1b-fb20-4769-85b6-655fa1efb118  ethernet  encbd10
```

2. Add the RoCE Express based interfaces to the bonded interface.

```
# nmcli con add type ethernet ifname eno5968 master mybond0
Connection 'bond-slave-eno5968' (b0bab623-b7b7-44ca-b01c-80fe8a97b708) successfully added.
# nmcli con add type ethernet ifname eno6032 master mybond0
Connection 'bond-slave-eno6032' (ad2fec5a-2c8b-42f6-8e16-37902d54b536) successfully added.
```

3. Confirm that four interfaces are now part of the bonded interface.

```
# nmcli c
NAME                  UUID                                  TYPE      DEVICE
bond-mybond0          1f543303-4594-4230-9d2f-45a5eb424305  bond      mybond0
bond-slave-encbd00    30127166-31d8-4004-a55f-981df4d09573  ethernet  encbd00
bond-slave-encbd10    6ae85d1b-fb20-4769-85b6-655fa1efb118  ethernet  encbd10
bond-slave-eno5968    b0bab623-b7b7-44ca-b01c-80fe8a97b708  ethernet  eno5968
bond-slave-eno6032    7533fcce-6da0-4239-9297-26b5e7a79e5c  ethernet  eno6032
```

4. Remove the OSA-Express-based interfaces from the bonded interface.

```
# nmcli con delete bond-slave-encbd00
Connection 'bond-slave-encbd00' (30127166-31d8-4004-a55f-981df4d09573) successfully deleted.
# nmcli con delete bond-slave-encbd10
Connection 'bond-slave-encbd10' (6ae85d1b-fb20-4769-85b6-655fa1efb118) successfully deleted.
```

5. Confirm that only the RoCE Express based interfaces remain as part of the bonded interface.

```
# nmcli c
NAME                  UUID                                  TYPE      DEVICE
bond-mybond0          1f543303-4594-4230-9d2f-45a5eb424305  bond      mybond0
bond-slave-eno5968    b0bab623-b7b7-44ca-b01c-80fe8a97b708  ethernet  eno5968
bond-slave-eno6032    7533fcce-6da0-4239-9297-26b5e7a79e5c  ethernet  eno6032
```

Result

Network traffic through `mybond0` now uses the RoCE Express PCI functions.

Migrating to UID-based network interface names

Adapt a network configuration for RoCE Express PCI functions to the predictable network-interface naming-scheme.

Goal

After a Linux upgrade, adapt an existing network configuration with RoCE Express to a new interface naming-scheme, see "Network interface names" on page 8.

Assumptions

Before the kernel upgrade, a RoCE Express based network interface has the name `enP4272p0s0`. NetworkManager, the network manager of the distribution, uses this name for a connection `con1`. The corresponding PCI function has the function address `10b0:00:00.0` and function ID `3e1`.

Strategy

Compute the predictable interface name and configure it alongside the existing interface before the kernel upgrade.

Sample procedure

Unless your distribution offers a migration tool that can handle changed network interface names, proceed according to the steps that follow.

1. Find out whether UID uniqueness checking is enabled in the environment where your Linux instance runs.

   Read the `uid_is_unique` attribute for the PCI function in the pci branch of sysfs.

   ```
   # cat /sys/bus/pci/devices/10b0:00:00.0/uid_is_unique
   1
   ```

   The predictable interface name depends on this information.

   1

   UID uniqueness checking is enabled. The interface name is based on the UID.

   a. Read the UID from sysfs.

   ```
   # cat /sys/bus/pci/devices/10b0:00:00.0/uid
   0x10b0
   ```

   b. Convert the hexadecimal value into a decimal number: `0x10b0 = 4272`. This decimal number is also used in `enP4272p0s0`. This match is common but not guaranteed. Always compute the decimal number.

   c. The network interface name is the `eno` prefix followed by UID in decimal notation: `eno4272`.

   0

   UID uniqueness checking not enabled. The interface name is based on the FID.

   a. Read the FID from sysfs.

   ```
   # cat /sys/bus/pci/devices/10b0:00:00.0/function_id
   0x03e1
   ```

   b. Convert the hexadecimal value into a decimal number: `0x03e1 = 993`.

   c. The network interface name is the `ens` prefix followed by FID in decimal notation: `ens993`.

   The following steps assume that UID uniqueness checking is enabled and so the interface name is `eno4272`.

2. Proceed according to how you are connected to the Linux instance.

   - If you are connected through a connection other than `con1` for `enP4272p0s0`, modify `con1` to use `eno4272`.

     Modifying a connection is disruptive, so it must not be busy. In particular, do not modify a connection while using it for a login session. Issue an **`nmcli`** command to modify the connection.

   ```
   # nmcli connection modify con1 connection.interface-name eno4272
   ```

- If you are connected through connection `con1` for `enP4272p0s0`, issue a **`nmcli`** command to add a connection for `eno4272`.

  ```
  # nmcli connection add con2 connection.interface-name eno4272 ...
  ```

  See the **`nmcli`** man page for further options.

  Change any specifications that use the connection name of the existing connection for interface name `enP4272p0s0` to use the name of the newly created connection for `eno4272`. For example, the old connection name might be used as part of custom scripts for system automation.

  Change any specifications, for example in custom scripts, to use the new interface name `eno4272` instead of the former `enP4272p0s0`.

3. When the upgraded Linux instance is booted, the connection for `eno4272` is used.

   You can use the **`ip`** command to confirm that the interface with the new name is active.

   ```
   # ip a
   1: lo: ...
      ...
   2: eno4272: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen
   1000
      link/ether 82:11:a1:b9:82:06 brd ff:ff:ff:ff:ff:ff
      altname ens993
      altname enP4272p0s0
      ...
   ```

   The output might include lines that begin with `altname` and specify the interface name according to other naming schemes. Although Linux can detect the mapping, do not rely on any network manager to use this information for automatically accommodating changed interface names.

4. If you have created a new connection for the new interface name, you can now delete the old connection.

Chapter 5. Scenario: Setting up Linux for SMC-R

To an existing OSA-based network connection, add a connection through a RoCE Express adapter and configure the network interfaces for SMC-R traffic.

Before you begin

Confirm that both software and hardware have sufficient levels to run SMC-R. On your Linux instance, issue an **smcr info** command.

```
# smcr info
Kernel Capabilities
SMC Version:      2.0
SMC Hostname:     a3545024.lnxne.boe
SMC-D Features:   v1 v2
SMC-R Features:   v1 v2

Hardware Capabilities
SEID:             n/a
ISM:              n/a
RoCE:             v1 v2
```

In the command output, The SMC-R and RoCE lines must list one or more version numbers. If n/a is shown instead of version information, SMC-R is not supported.

About this task

The scenario assumes that a Linux instance runs in a z/VM guest virtual machine with z/VM user ID A3545024. The z/VM administrator already enabled z/VM for PCI usage and made the PCI function with FID 110e available to the Linux instance. In the IOCDS, the hardware definition assigns PNET ID NET10 to the PCI function. The PCI function connects to an external LAN.

Through a z/VM VSWITCH and an OSA-Express adapter, the Linux instance has a TCP/IP connection to the same external LAN as the PCI function. The OSA interface on Linux is assumed to be encbdf0.

The goal is to add another connection for SMC-R traffic through the PCI function of the RoCE Express adapter.

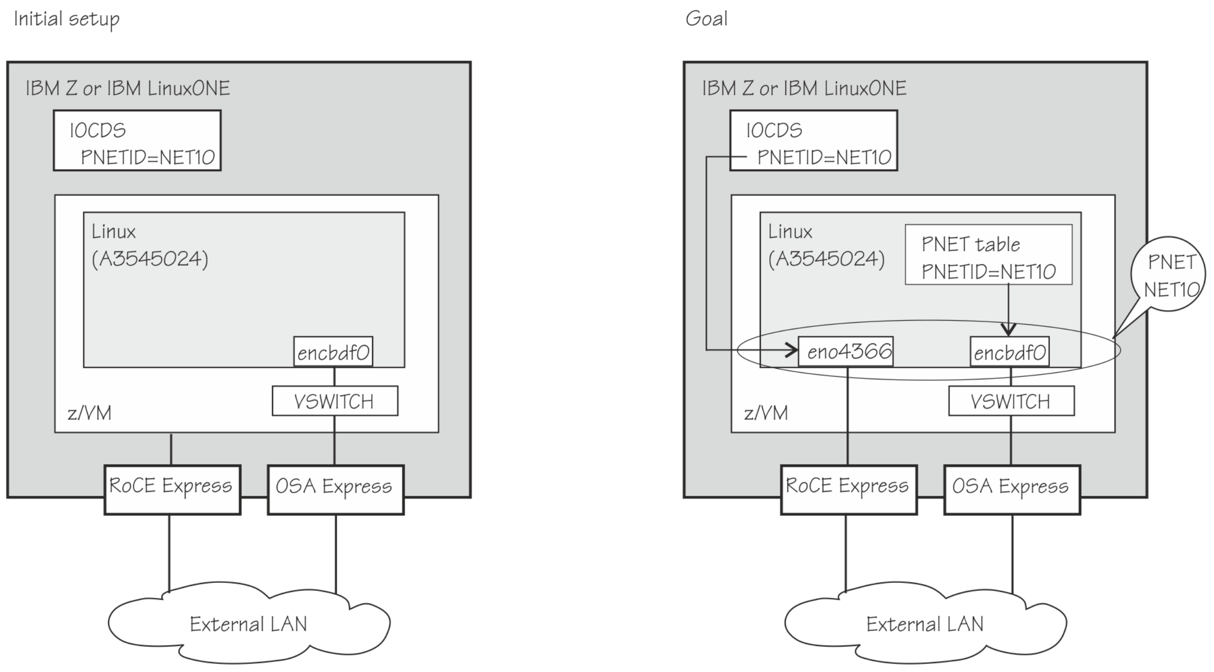


Figure 6. Initial setup and goal

Procedure

Perform the following steps from Linux.

1. List the available PCI functions.

```
# vmcp 'query pcifunction'
PCIF 0000110E FREE                         DISABLED RoCE Express 2
```

   The DISABLED in the output indicates that the PCI function is not yet attached to the z/VM guest virtual machine.

2. Attach the PCI function.

```
# vmcp 'attach pcifunction 0000110E to *'
PCI FUNCTION 0000110E ATTACHED TO A3545024 0000110E
```

   Repeating the query command now shows the PCI function as attached to A3545024.

```
# vmcp 'query pcifunction'
PCIF 0000110E ATTACHED TO A3545024 0000110E ENABLED  RoCE Express 2
```

3. Display information for the PCI function.

```
# smc_rnics
    FID  Power  PCI_ID        PCHID  Type           PPrt  PNET_ID           Net-Dev
-----------------------------------------------------------------------------------------
   110e  1      110e:00:00.0  020c   RoCE_Express2  0     NET10             eno4366
```

   The command output shows `NET10` as the PNET ID that the hardware configuration assigns to the PCI function.

4. Create a link with an IP address for the PCI function.

   This link is required for the SMC-R setup but it must not be used for any IP traffic. The following **`ip`** commands set up a suitable but transient link for our example. For a persistent setup, use the network manager of your distribution.

```
# ip link add dev mylink link eno4366
# ip addr add 192.0.2.1/24 dev mylink
# ip link set mylink up
# ip route flush scope link dev mylink
```

   The final command removes all possible routes for this link and so prevents any traffic. The network manager of your distribution might interpret this link configuration as erroneous and take corrective actions. To preserve the configuration, remove the interface from the scope of the network manager. For example, to stop NetworkManger from changing the link configuration, issue the following **`nmcli`** command:

```
# nmcli device set eno4366 managed no
```

5. Assign the PNET ID of the PCI function to the network interface that the z/VM VSWITCH provides to the guest.

   Because this interface is a virtual network component, the PNET ID cannot be set in the hardware configuration. Therefore, use the **`smc_pnet`** command to create an entry in a PNET table.

```
# smc_pnet --add NET10 --interface encbdf0
```

6. Confirm your setup with the **`smc_chk`** command.

   - If a peer with PNET ID NET10 is already set up for an SMC-R connection, you can test the connection to the peer.

     The following command confirms SMC-R connectivity to a peer, for example, an SMC-R capable service on a z/OS instance with IP address 192.168.5.47 and port 23.

```
# smc_chk -C 192.168.5.47 -p 23
  Live test (SMC-D and SMC-R)
    Success, using SMC-R
```

- Alternatively, you can confirm the local setup through a loopback connection to a local server application.

  In the following example, the command starts a local server on port 37373.

```
# smc_chk -S &
Server started on port 37373
```

  The following command confirms that the server can be reached through an SMC-R connection.

```
# smc_chk -C 127.0.0.1 -p 37373
Test with target IP 127.0.0.1 and port 37373
  Live test (SMC-D and SMC-R)
     Success, using SMC-R
```

  The following command stops the server.

```
# killall smc_chk
```

Results

The Linux instance is now ready for SMC-R connections.

What to do next

Use the **`smcr stats`** command to observe traffic patterns and for problem determination if SMC-R connections cannot be established. Use the **`smcss`** command to monitor individual connections.

Accessibility

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

Documentation accessibility

The Linux on IBM Z and LinuxONE publications are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF file and want to request a Web-based format for this publication send an email to eservdoc@de.ibm.com or write to:

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany

In the request, be sure to include the publication number and title.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility at

```
www.ibm.com/able
```

Notices

Trademarks

SC34-7745-00