Linux on IBM Z and IBM LinuxONE

SMC-D via ISM pass-through performance evaluation for KVM guests on IBM® Z®

About this publication

Before using this information and the product it supports, please read the information in chapter 11 Notices and disclaimer on page 31.

The following study compares SMC-D via ISM pass-through for KVM guests to HiperSockets connectivity that uses MacVTap. Both networking solutions apply to KVM-guest-to-LPAR communication within a single CPC. Comparison metrics include throughput and processor consumption.
A separate chapter deals with tuning of buffer sizes for SMC-D and its influence on performance. Furthermore, it is explained how to set up SMC-D via ISM pass-through and what needs to be considered.

Authors: Jens Markwardt, Nils Hoppmann

Table of contents

List of figures

1 Abstract

This study compares two network interconnect solutions for KVM guests within a Central Processor Complex (CPC). The conventional HiperSockets technology is compared with a new approach that utilizes pass-through of Internal Shared Memory (ISM) devices exploiting the Shared Memory Communications – Direct Memory Access (SMC-D) protocol. The evaluation is performed on an IBM® z16™ running Red Hat® Enterprise Linux® (RHEL) 8.8 for IBM Z®.
Both networking solutions are compared in terms of latency, throughput and processor consumption using the *uperf* network benchmark. *Uperf* scenarios include a request-response and a streaming workload. Furthermore, the impact of tuning buffer sizes on the performance of SMC-D workloads is discussed as a separate topic.
Results demonstrate that SMC-D via ISM pass-through for KVM guests (definition provided in chapter 3.3) exhibits superior performance characteristics. It clearly outperforms HiperSockets connectivity via MacVTap in terms of latency and throughput as well as processor consumption.

2 Introduction

The focus of this study for IBM® Z® and IBM® LinuxONE is on network communication between a Linux system as KVM guest on a logical partition (LPAR) and another Linux system on (bare-metal) LPAR. For ease of reading this configuration is subsequently called *guest-to-LPAR communication*. Hereby, both LPARs reside within a single CPC.

Current networking solutions for guest-to-LPAR communication within a CPC are typically based on HiperSockets technology. In those setups, the attachment of HiperSockets to the KVM guest is often accomplished using virtio networking along with the vhost-net host device driver in MacVTap mode. This solution is easy to set up and does not require any additional hardware or cabling. However, despite being tuned for optimal performance the vhost-net/virtio-net architecture for KVM guests still adds considerable overhead compared to direct-attached network adapters within LPAR installations. This overhead is often visible as increased processor consumption of the KVM host, lower networking throughput, and increased latency.

This study introduces SMC-D via ISM pass-through for KVM guests as an additional networking option for guest-to-LPAR communication within a CPC. SMC-D is based on ISM technology and uses virtual PCI network adapters. These are directly attached to a KVM guest via PCI pass-through and thus eliminate the overhead introduced by the vhost-net/virtio-net architecture.
For the sake of completeness, it should be mentioned that SMC-D via ISM pass-through connectivity can be utilized not only for guest-to-LPAR communication, but also for communication between KVM guests within a CPC. Hereby, the guests can either reside in the same KVM host or in different LPARs. However, the focus here is solely on guest-to-LPAR communication. Of course, SMC-D can as well be used for communication between LPARs.

SMC-D via ISM pass-through is available for IBM z15 and z16 with the ISM feature enabled. A list of supported Linux on IBM Z distributions is provided in chapter 3.2.

2.1 How this paper is structured

Chapter 3 explains what SMC-D via ISM pass-through stands for and provides a technical overview of related terms and definitions. It also briefly discusses different SMC flavors and versions as well as the distinction between SMC-D via ISM and TCP/IP via HiperSockets.

Chapter 4 introduces *uperf* as the network benchmark used in this study. It further describes the applied workload patterns and processor consumption metrics.

Chapter 5 discusses the individual components of the test environment. This includes a detailed configuration description of the logical partitions involved, the operating system and the KVM guest setup. Hereby, special focus is on network connectivity.

Readers who wish to execute the benchmark tests mentioned in this study themselves will find instructions on how to reproduce the test environment in chapter 6. Instructions for running *uperf* can be found here as well.

Finally, all measurement results including processor consumption are presented in chapter 7.
A separate subsection treats the impact of buffer-size tuning on workloads that use SMC-D.

The study is concluded with takeaways in chapter 8.

3 About SMC-D via ISM pass-through

This chapter provides a brief overview of the idea behind SMC-D via ISM pass-through, its basic principles and the related terms and definitions in the context of this study.

3.1 Shared Memory Communications (SMC)

Before addressing SMC-D via ISM pass-through, this chapter will first outline the fundamentals and variants of the underlying SMC protocol.

3.1.1 SMC-D and SMC-R

The SMC-D protocol is one form of shared memory communication that exploits ISM technology. ISM is a virtual PCI adapter that enables direct access to shared virtual memory across LPARs within a single CPC. For SMC-D frequently asked questions (FAQ), see [1].
The other shared memory communication protocol is Shared Memory Communications – Remote Direct Memory Access (SMC-R). It makes use of the Remote Direct Memory Access (RDMA) capabilities of RoCE Express adapters. Thus, SMC-R is primarily used for communication between instances on different CPCs.
For communication within the same CPC, the use of SMC-D via ISM has a cost benefit of not requiring any physical cabling or additional physical network devices. In this respect SMC-D via ISM is similar to HiperSockets.

3.1.2 HiperSockets traffic and SMC-D via ISM

It is important to note that one cannot blindly replace HiperSockets traffic with SMC-D via ISM. One major point here is that the SMC protocol is based on TCP. As a consequence, non-TCP based network traffic - like UDP traffic - cannot use SMC. TCP also remains the fallback mechanism in case SMC connection negotiation fails. Due to these requirements, SMC-D via ISM supplements HiperSockets but does not replace them.

3.1.3 SMC version 2 and SMC-D version 2

SMC version 1 (SMCv1) only provides single IP subnet support, while SMC version 2 (SMCv2) provides additional layer 3 support, thus enabling connectivity spanning over multiple IP subnets. Note that the SMC protocol applies to both SMC-D and SMC-R. For in-depth technical information about SMCv2, see [2].
Besides other extensions to SMC-D version 1 (SMC-Dv1), SMC-D version 2 (SMC-Dv2) allows the use of non-native ISM devices. These are discussed in chapter 6.2.1.2 and beneficial for use with virtio-net devices in a KVM guest.

SMC-Dv2 requires Internal Shared Memory version 2 (ISMv2) virtual PCI functions. The hardware requirements are as follows:

- IBM z15 or z16 with ISMv2 feature: T01 MCL P46601.067 and T02 base

SMC-Dv2 is available for the following (or later) Linux on IBM Z distributions:

- Red Hat® Enterprise Linux® 8.4
- SUSE Linux® Enterprise Server 15 SP3
- Ubuntu Server 21.04

Although the focus of this paper is on Linux, it should be mentioned that SMC-Dv2 is also available for the IBM z/OS® operating system. Software requirements are:

- IBM z/OS® 2.4 SMCv2 enablement software: OA59152/UJ03768 and PH22695/UI71143

3.2 PCI pass-through

ISM devices are represented as virtual PCI adapters that can be attached to virtual machines via PCI pass-through. The PCI pass-through feature allows a virtual machine direct access to a PCI device function. It appears as if the PCI device was physically attached to the virtual machine. Thus, the virtual machine does not see any kind of virtualized device, but the real PCI device.
PCI device pass-through for KVM guests is available for the following (or later) Linux on IBM Z distributions:

- Red Hat® Enterprise Linux® 8.8 and 9.2
- SUSE Linux® Enterprise Server 15 SP5
- Ubuntu Server 23.04

3.3 Definition of terms “MacVTap HiperSockets” and “SMC-D via ISM pass-through”

In the context of this study the following two terms are introduced for ease of reading:

1. *MacVTap HiperSockets* is a setup where a HiperSockets adapter is attached to a KVM guest via virtio networking using the vhost-net host device driver in MacVTap mode.
2. *SMC-D via ISM pass-through* is a colloquial term meaning pass-through of an ISM virtual PCI function to a virtual machine (e.g. a KVM guest) for use with the SMC-D protocol.

In the following, whenever referring to the term *MacVTap HiperSockets*, it is assumed that the TCP/IP protocol is used.

3.4 SMC rendezvous

SMC-D requires an IP network and existing TCP/IP connectivity for initial connection setup. This implies that both peers must be able to reach each other through TCP/IP. The connection setup process consists of the TCP 3-way handshake and an additional SMC specific connection layer control (CLC) handshaking step, that is described in chapter 6.2.1.3. This whole procedure is called *SMC rendezvous*.
After the connection has been established, the actual data transfer is accomplished via DMA utilizing the ISM adapter. Technically, the SMC protocol layer is transparently inserted as an “out of band” communications layer under TCP/IP. The TCP/IP connection also serves as a possible fallback mechanism if the SMC-D connection setup should fail.

When using SMC-D on LPAR, the IP network required for the SMC rendezvous can be provided either via physical devices – by using OSA-Express or RoCE Express adapters – or via HiperSockets. To avoid the use of physical adapters, the preferred method is HiperSockets.

For KVM guests any virtio-net device can be used for rendezvous processing. Thus, it is possible to re-use existing MacVTap HiperSockets to perform the SMC rendezvous for SMC-D. Alternatively, a RoCE Express adapter can be used as a pass-through PCI device for the same purpose. In cases where virtio-net devices are utilized for rendezvous processing, non-native ISMv2 devices are a natural choice if SMC-Dv2 is available. See chapter 6.2.1.2 for further explanations on this topic.

4 Workload

The network benchmark used is a slightly adapted version of *uperf*.

Uperf can be run either in client or server mode. Furthermore, a workload description in the form of an XML file is required for its operation. This profile – for simplicity from now on called `profile.xml` – resides on the *uperf* client. It contains the characteristics of the workload like sizes of read and write messages, number of parallel connections, runtime as well as additional meta

information such as the IP address of the *uperf* server and the network protocol. An example is provided in appendix 9.2.

The code snippet below shows how *uperf* can be invoked in server and client mode:

```
server# uperf -s
client# uperf -a -i 30 -m profile.xml
```

The benchmark is started in server mode using the '`-s`' parameter. On the client side, *uperf* is called with a set of parameters. The statement from above will cause *uperf* to gather all available statistics ('`-a`') and print throughput metrics every 30 seconds ('`-i 30`') while running the workload defined in the profile ('`-m profile.xml`'). See http://uperf.org/ and https://github.com/uperf/uperf for more information.

4.1 Workload profiles

In this study the focus is on two different *uperf* workload patterns, *rr1c-200x1000* and *str-readx30k*.
The rr1c-200x1000 pattern is a request-response workload simulating highly transactional and latency-critical traffic with medium data sizes (like web workloads). A client sends a request of 200 bytes and receives a 1000-byte response from the server. The focus is on transaction duration, i.e. how fast data can be transferred instead of how much data. Lower transaction times are favored, since they reflect a faster transmission.
The str-readx30k pattern is a streaming workload where the *uperf* server permanently sends data in 30720-byte chunks. The *uperf* client receives and acknowledges this data. This setup can mimic a restore step of a backup operation. In this pattern, the focus is on throughput (Gb/s). Thus, higher numbers are of interest.
The execution of each of the two workload patterns is repeated with 1, 10, 50 and 250 active parallel connections. The runtime of a single measurement – with a fixed workload pattern and a fixed number of parallel connections – is 300 seconds.

4.2 Processor consumption

The processor consumption is measured in terms of how much CPU time was spent to transfer a certain amount of data. Thus, the time is indicated in CPU microseconds (µs) per transferred Gigabit (Gb). The processor consumption for the *uperf* client and server systems can potentially differ a lot, especially with asymmetric workloads.
This study focuses on processor consumption of the *uperf* server only. Within the test environment, the *uperf* server runs in a KVM guest (see chapter 5.2). The measured processor consumption comprises the entire LPAR and includes both the KVM host and the KVM guest. This allows for a direct comparison of the CPU consumption overhead that is introduced by MacVTap HiperSockets versus SMC-D via ISM pass-through. The processor consumption of the *uperf* client – that runs on bare-metal LPAR – is not analyzed.
Detailed information on the math behind *uperf* throughput and processor consumption calculations can be found in [3].

5 Test environment

This chapter describes the setup of the test environment. It includes LPAR, operating system and KVM guest configuration. Particular attention is paid to network connectivity setup.

5.1 LPAR configuration

The test environment consists of two LPARs located in a single IBM z16 CPC. Both LPARs are defined with the same characteristics:

- Microcode level: Driver D51C, Bundle S12
- Number of cores (Integrated Facilities for Linux): 4
- Number of SMT-2 threads / CPUs: 8
- Memory: 32 GiB
- Network: HiperSockets, 32K MTU size, ISMv2 device, PNET ID = none

The test environment uses only layer 2 connectivity.

5.2 Linux and KVM host configuration

The setup comprises three instances of RHEL 8.8 based on Linux kernel version 4.18.0-477.10.1.el8_8.s390x.

On the KVM host, the following additional packages provided by RHEL 8.8 are used:

- qemu-kvm-6.2.0-32
- libvirt-8.0.0-19

There are two instances of RHEL 8.8 on LPAR A – KVM host and a single KVM guest – and a third instance on LPAR B as figure 1 illustrates. It should be noted that figure 1 only shows the base setup without any network connectivity.

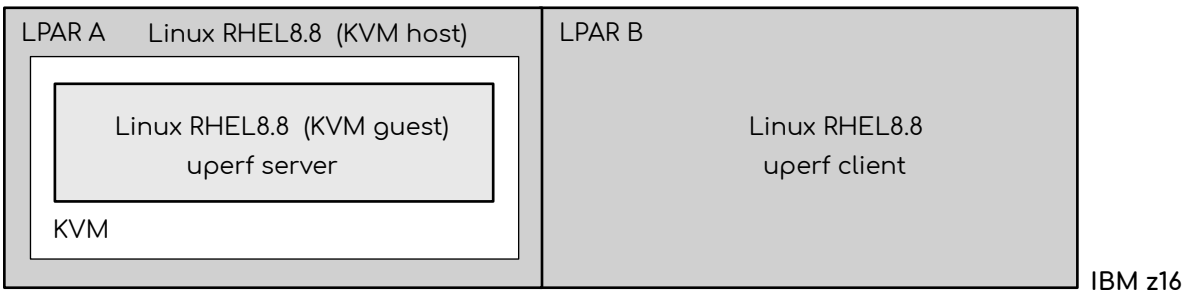

Figure 1: Linux and KVM host configuration w/o network connectivity

During benchmark execution LPAR B takes the role of the client whereas the KVM guest on LPAR A acts as the server.

5.3 KVM guest configuration

The KVM guest is set up as follows:

- Number of virtual CPUs: 8
- Memory: 4 GiB
- Network: MacVTap HiperSockets,
  Pass-through ISMv2 device

In the test setup, the MacVTap HiperSockets adapter uses the vhost-net host device driver in MacVTap mode with one rx/tx queue. The corresponding MacVTap interface domain xml is discussed in chapter 6.1.3.

5.4 MacVTap HiperSockets measurement setup

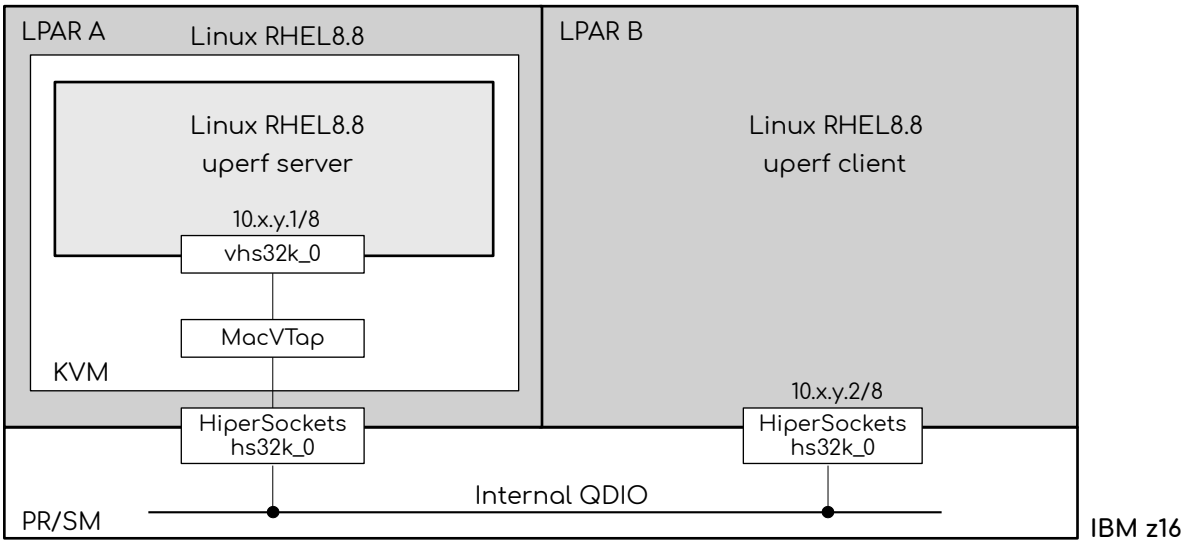

Figure 2: MacVTap HiperSockets measurement setup

Figure 2 depicts a typical setup for a guest-to-LPAR communication based on HiperSockets. On LPAR A the HiperSockets adapter is attached to the KVM guest via virtio networking in MacVTap mode. In this study, this setup serves as the reference for subsequent measurements. The HiperSockets implementation is based on the Queued Direct I/O (QDIO) protocol. For details see the IBM HiperSockets Implementation Guide [4].

5.5 SMC-D via ISM pass-through measurement setup

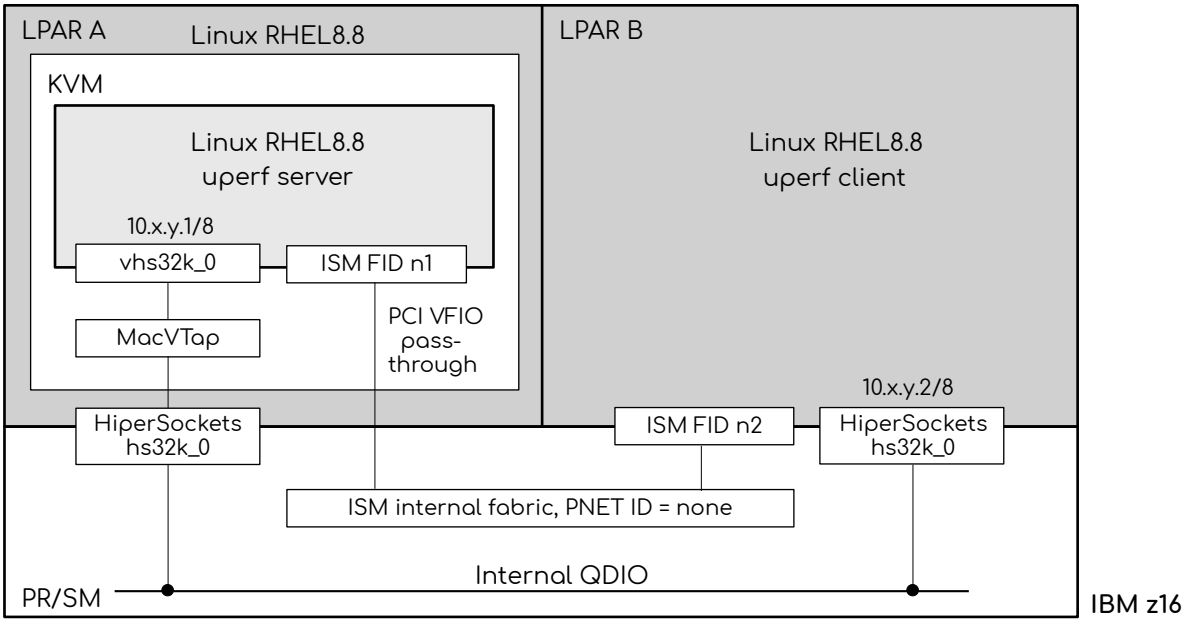

Figure 3: SMC-D via ISM pass-through measurement setup

Figure 3 illustrates the proposed SMC-D setup. It is based on PCI pass-through of a virtual ISM PCI function (ISM FID) to the KVM guest, thereby enabling it for SMC-D communication.
In this setup the MacVTap HiperSockets connectivity is only used to perform the SMC rendezvous (see chapter 3.4). The actual data transfer is carried out via SMC-D using the internal ISM fabric.
The terms *FID* and *PNET ID* mentioned in figure 3 are explained in chapter 6.2.1.

It should be pointed out that the described SMC-D setup just adds an ISM device with pass-through functionality. The existing HiperSockets setup is not affected by this and remains fully functional. Eventually, the operator of the environment can decide whether to use MacVTap HiperSockets or SMC-D via ISM pass-through.

For information on other KVM-related networking options to tune KVM host and KVM guest environments, see [9].

6 Setup guidance

This chapter provides some guidance for setting up MacVTap HiperSockets and ISM pass-through. It is further explained how to invoke the *uperf* benchmark for SMC-D workloads.

6.1 Setting up MacVTap HiperSockets

Before using MacVTap HiperSockets, some prior configuration steps are required in the Input/Output Configuration Data Set (IOCDS) as well as on the KVM host.

6.1.1 HiperSockets definition in the IOCDS

HiperSockets are defined in the IOCDS. The test environment of this study uses HiperSockets with an MTU size of 32 KiB and a frame size of 40 KiB. For details see [4], chapter 2: “HiperSockets environment definitions”.

6.1.2 Setting up HiperSockets in layer 2 mode and increasing the buffer count

On the KVM host, the HiperSockets adapter needs to be configured in layer 2 mode to make it work properly as a virtio device in the KVM guest.
For performance reasons the buffer count is increased from 64 to 128. The example below shows the required steps using *sysfs* for a HiperSockets adapter with channel ID 0.0.8200:

```
# Set device offline
echo 0 > /sys/devices/qeth/0.0.8200/online
# Set layer 2 mode
echo 1 > /sys/devices/qeth/0.0.8200/layer2
# Increase buffer count to 128
echo 128  > /sys/devices/qeth/0.0.8200/buffer_count
# Set device online again
echo 1 > /sys/devices/qeth/0.0.8200/online
```

The settings can be verified using the *lsqeth* command (requires package *qethconf*).
See appendix 9.1 for an *lsqeth* example output for a HiperSockets adapter in layer 2 mode and a buffer count of 128.
Persistent changes of device attributes can be accomplished using the *chzdev* command. Examples can be found in [5]: Setting the layer2 attribute and Specifying the number of inbound buffers. For details about *lsqeth* see lsqeth - List qeth-based network devices.

6.1.3 MacVTap interface configuration

A MacVTap interface is defined in the guest domain xml on the KVM host. The procedure is described in [6]: Configuring a MacVTap interface as well as in [7]. The latter also contains additional performance considerations.
The test environment uses the following MacVTap definition which needs to be placed into the interface section of the domain xml:

```
<interface type="direct" dev="vhs32k_0">
  <mac address="12:34:56:78:9a:bc"/>
  <source dev="hs32k_0" mode="bridge"/>
  <model type="virtio"/>
  <driver name="vhost"/>
</interface>
```

Note that the device names and the mac address must be adjusted accordingly.

6.2 Setting up ISM pass-through to a KVM guest

Setting up pass-through of an ISM virtual PCI function to a KVM guest include defining the ISM device in the IOCDS and setting up the KVM host for PCI device pass-through to the guest.

6.2.1 ISM device definition

ISM devices are defined in the IOCDS. Since ISM is a virtual (firmware) device it is provided with a virtual channel ID (VCHID). For further details on ISM VCHIDs, see [2].
Each ISM device is configured with multiple virtual functions (VFs). These show up as virtual PCI devices and are identified by a unique function ID (FID).
ISM FIDs can be assigned to LPARs or guests and hereby provide similar conceptual functions as single-root input/output virtualization (SR-IOV). Be aware that each individual FID can only be assigned to one single LPAR or guest. Thus, for multi-guest scenarios each guest requires its own FID.
It should be noted that two peers can only communicate via SMC-D if their FIDs belong to the same ISM device (same VCHID).

6.2.1.1 UID

For PCI pass-through to a KVM guest, the PCI device to be passed through is selected by its PCI address. However, PCI addresses may change across reboots and when additional FIDs are assigned to (or removed from) an LPAR. Persistent PCI addresses can be achieved by defining a unique ID (UID) for each FID along with the activation of unique user-defined identifier (UUID) checking for that LPAR in the IOCDS.
The described test environment uses a configuration where the UID equals the FID.
Note that the UID also determines the PCI domain of a virtual function.

6.2.1.2 PNET ID, native and non-native ISM devices

The physical network ID (PNET ID) is set in the IOCDS and used to associate an ISM device with an Ethernet device such as a RoCE Express, OSA-Express or HiperSockets adapter to perform the SMC rendezvous. In that context an ISM device and an Ethernet device are called *associated* if they have the same PNET ID. An ISM device with a PNET ID set is called a *native* ISM device. Note that in a native ISM setup, both peers must have the same PNET ID for SMC-D communication to work. The native ISM setup is supported with both SMC-Dv1 and SMC-Dv2. For more information about device association using PNET IDs, see [5]: Internal shared memory device driver.

In contrast to SMCv1, SMCv2 allows to perform the SMC rendezvous for ISMv2 devices that do not have a PNET ID set. Those are called *non-native* ISMv2 devices and do not have an associated Ethernet device. In a non-native ISMv2 setup, none of the peers may have a PNET ID set.

The use of non-native ISM devices is beneficial if the SMC rendezvous is performed using virtio-net interfaces (e.g. MacVTap HiperSockets). This is because virtio-net interfaces cannot be assigned a PNET ID in the IOCDS since they only exist while the KVM guest is running. Even though it is possible to manually assign a PNET ID to an Ethernet interface from within Linux using the *smc_pnet* command (see [5]: smc_pnet - Create network mapping table) it is more convenient to use non-native ISMv2 devices instead.

6.2.1.3 Excursion: CLC handshake for SMC

This subsection is a brief digression into the topic of CLC handshake for SMC. It was deliberately placed here because all the necessary terms needed to be introduced first.

As already mentioned in chapter 3.4, the CLC handshake is part of the SMC rendezvous. During the CLC handshake, it is checked whether SMC-D or SMC-R can be utilized. If both options are available SMC-D is prioritized. If neither is possible, TCP is used as fallback. As the focus of this work is on SMC-D, it is assumed in the following section that no SMC-R/RDMA-capable devices are available.

As part of the CLC handshake procedure for SMC-D, both peers check whether a common ISM VCHID exists, as this is a prerequisite for the use of SMC-D. For that purpose one side proposes either a single VCHID or a list of VCHIDs to its peer on the other side:

- A single VCHID is proposed for native ISM devices that have already been associated via a PNET ID with the Ethernet device initiating handshake.
- A list of other available VCHIDs is proposed for non-native ISMv2 devices, since none of them is associated with the Ethernet device initiating the handshake.

After one or more VCHIDs have been proposed, the peer on the other side in turn evaluates all its available VCHIDs to determine whether any common VCHID exists. If a common VCHID is found during the CLC handshake, the common ISM device can be used as DMA device. Otherwise, TCP fallback mode is used.
Please note that the given description is only a very simplified representation of the actual handshake procedure. In reality, numerous additional tests are necessary before an ISM device can be used for DMA.

From a performance perspective, carrying out the CLC handshake takes some additional time. This results in a disadvantage of SMC-D for "short lived" connections compared to pure TCP/IP workloads. For example, if a workload only sends one request and then expects a short response, SMC-D simply has an overhead that it cannot make up for.

6.2.2 KVM host setup

This subsection provides detailed information about setting up a KVM host for PCI device pass-through to a KVM guest using ISM device pass-through as an example. It discusses the prerequisites, delivers instructions for creating the required domain xml entries and advises on how to achieve persistent PCI mappings.

6.2.2.1 Checking for vfio-pci kernel module

To use an ISM device for PCI pass-through it must be controlled by the *vfio-pci* device driver. The *modinfo* command can be used to verify that the kernel module is available and loaded:

```
kvmhost# modinfo vfio-pci
filename:       /lib/modules/4.18.0-477.10.1.el8_8.s390x/kernel/drivers/
                vfio/pci/vfio-pci.ko.xz
description:    VFIO PCI - User Level meta-driver
```

6.2.2.2 Useful software packages

The *smc-tools* package should be installed. It provides commands like *smc_rnics* and *smc_run:*

- *smc_rnics* lists RoCE Express adapters and ISM PCI functions.
- *smc_run* runs a TCP socket program using the SMC protocol.
  It is used to run the *uperf* benchmark in this study (see chapter 6.3.1).

For more information, see [5]: smc_rnics - list RoCE Express PCI functions and control their online state and smc_run - Run a TCP socket program with the SMC protocol using a preloaded library.
In addition, the *pciutils* package can be helpful for debugging purposes, as it provides the *lspci* command. *lspci* lists PCI devices attached to the system and their characteristics.

6.2.2.3 Determining the PCI address of the ISM device

The *smc_rnics* or *lspci* commands can be used to check whether the ISM device is present and to determine its PCI address. Note that *smc_rnics* (without arguments) outputs one line for each FID:

```
host# smc_rnics
FID  Power  PCI_ID        PCHID  Type           PPrt  PNET_ID            Net-Dev
------------------------------------------------------------------------------
605  1      0605:00:00.0  07c8   ISM            n/a   n/a                n/a

host# lspci -v
0605:00:00.0 Non-VGA unclassified device: IBM Internal Shared Memory (ISM)
virtual PCI device
    Physical Slot: 00000605
    Flags: bus master, fast devsel, latency 0, IOMMU group 0
    Memory at 4000000000000000 (64-bit, prefetchable) [virtual] [size=256T]
    Memory at 4001000000000000 (64-bit, prefetchable) [virtual] [size=256]
    Capabilities: [40] MSI: Enable+ Count=1/32 Maskable- 64bit+
    Kernel driver in use: ism
    Kernel modules: ism
```

A PCI address has the following format: `<domain>:<bus>:<slot>.<function>`. For more information about the meaning of the address component values for Linux on IBM Z, see [5]: PCI Express support.
Note that the PNET_ID reported by the *smc_rnics* command shows ‘`n/a`’ indicating that the ISM device has been defined in non-native mode.

6.2.2.4 Configuring PCI pass-through to a KVM guest

To enable PCI device pass-through to a KVM guest an additional element called `<hostdev...>` needs to be added into the `<devices...>` section of the KVM guest domain xml as shown below:

```
<hostdev mode="subsystem" type="pci" managed="yes">
  <source>
    <address domain="0x0605" bus="0x00" slot="0x00" function="0x0"/>
  </source>
  <address type="pci">
    <zpci uid="0x00605" fid="0x00000605"/>
  </address>
</hostdev>
```

The source address attribute elements (`domain`, `bus`, `slot`, `function`) need to match the PCI address as printed by the *lspci* or *smc_rnics* command on the KVM host as listed in 6.2.2.3.

Just like in the KVM host, the PCI address of the pass-through device in the KVM guest can change across reboots. The concept of UIDs for persistent PCI addresses, as discussed in chapter 6.2.1.1, also applies to KVM guests in the same way. The UID can be freely chosen and has to be entered using the `uid` attribute in the `<zpci...>` element of the guest domain xml. As with the KVM host before, the UID=FID setting is used here.

The PCI address as seen in the guest is solely determined by the `uid` attribute that becomes the PCI domain. All remaining elements of the PCI address (`bus`, `slot`, `function`) are set to zero.

For more information, see [6]: Preparing PCI pass-through devices and Configuring pass-through PCI devices. There is also a related “KVM on IBM Z and LinuxONE” blog post [8] describing more technical details.

When the KVM guest is started, the *lspci* or *smc_rnics* command can be used to verify that the ISM pass-through device is present and uses the *ism* kernel module:

```
guest# lspci -v
0605:00:00.0 Non-VGA unclassified device: IBM Internal Shared Memory (ISM)
virtual PCI device
      Physical Slot: 00000605
      Flags: bus master, fast devsel, latency 0, IRQ 12, IOMMU group 1
      Memory at 4001000000000000 (64-bit, prefetchable) [virtual] [size=256T]
      Memory at 4002000000000000 (64-bit, prefetchable) [virtual] [size=256]
      Capabilities: [40] MSI: Enable+ Count=1/32 Maskable- 64bit+
      Kernel driver in use: ism
      Kernel modules: ism

guest# smc_rnics
FID  Power  PCI_ID        PCHID  Type           PPrt  PNET_ID            Net-Dev
------------------------------------------------------------------------------
605  1      0605:00:00.0  07c8   ISM            n/a   n/a                n/a
```

6.3 Enabling workloads for SMC-D (by the example of *uperf*)

To use *uperf* with SMC-D workloads, it is generally not necessary to adapt the `profile.xml`. The IP address to be specified for the *uperf* server in the `profile.xml` is the address of the TCP interface that is used for SMC rendezvous processing. Remember that SMC can only be used with TCP workload profiles.

6.3.1 Options to enable SMC-D

The SMC network protocol can be used in three ways:

1. Use of the *smc_run* command as a prefix to the TCP socket program:

   ```
   smc_run <program> <program_parameters>
   ```

2. Definition of `libsmc_preload` as a preload library before running the TCP socket program:

   ```
   export LD_PRELOAD=$LD_PRELOAD:libsmc_preload.so;
   <program> <program_parameters>
   ```

   Note that `LD_PRELOAD` can also be set globally in the system-wide configuration file `/etc/environment`.

3. Adjustment of socket calls in the program code by replacing the address family type *AF_INET* by *AF_SMC*.

Under the cover, all three variants eventually only change the address family type to *AF_SMC*.

To run *uperf*, the test environment uses the first and simplest method – the *smc_run* command – as shown below:

1. Start the *uperf* server in the KVM guest on LPAR A:

   ```
   server# smc_run uperf -s
   ```

2. Start the *uperf* client on LPAR B:

   ```
   client# smc_run uperf -a -i 30 -m profile.xml
   ```

For more information about the *smc_run* command, see [5]: smc_run - Run a TCP socket program with the SMC protocol using a preloaded library.

6.3.2 SMC-D connection statistics

The *smcd* command is also part of the *smc-tools* package and can be used to display various connection statistics for SMC-D.
Part of the output contains information about whether an SMC connection was established successfully or whether the connection used TCP instead (fallback mode).

The example below shows a snippet for a medium-size *uperf* request-response workload with 50 parallel connections. The line labeled *TCP fallback* reports cases where the underlying ISM device was not used and the data transfer was accomplished with TCP. Note that there is one extra connection being opened and used by *uperf* for management purposes. Thus, a total of 51 connections is reported:

```
client# smcd -d stats
SMC-D Connections Summary
  Total connections handled            51
  SMC connections                      51 (client 51, server 0)
    v1                                  0
    v2                                 51
  Handshake errors                      0 (client 0, server 0)
  Avg requests per SMC conn      14046725.0
  TCP fallback                          0 (client 0, server 0)
```

The output indicates that all (51) connections used SMC-Dv2 and none of them went into TCP fallback mode. For the sake of completeness, the full output is listed in appendix 9.3. For more information, see [5]: smcd - Display information about SMC-D link groups and devices.

7 Results

Measurement results are presented in this chapter. In the first two subsections, SMC-D via ISM pass-through performance is compared with MacVTap HiperSockets for the request-response and the streaming workloads stated in chapter 4.1. A third subsection deals with tuning of buffer sizes for SMC-D and its influence on performance.

7.1 Request-response workload

Figure 4 shows a comparison of transaction times between MacVTap HiperSockets and SMC-D via ISM pass-through for the medium-size request-response workload (rr1c-200x1000). The measurements were repeated four times with different numbers of active connections, i.e. 1, 10, 50 and 250 parallel connections. The transaction times shown are normalized to the single-connection MacVTap HiperSockets case. Shorter bars correspond to lower transaction times and are hence preferred.

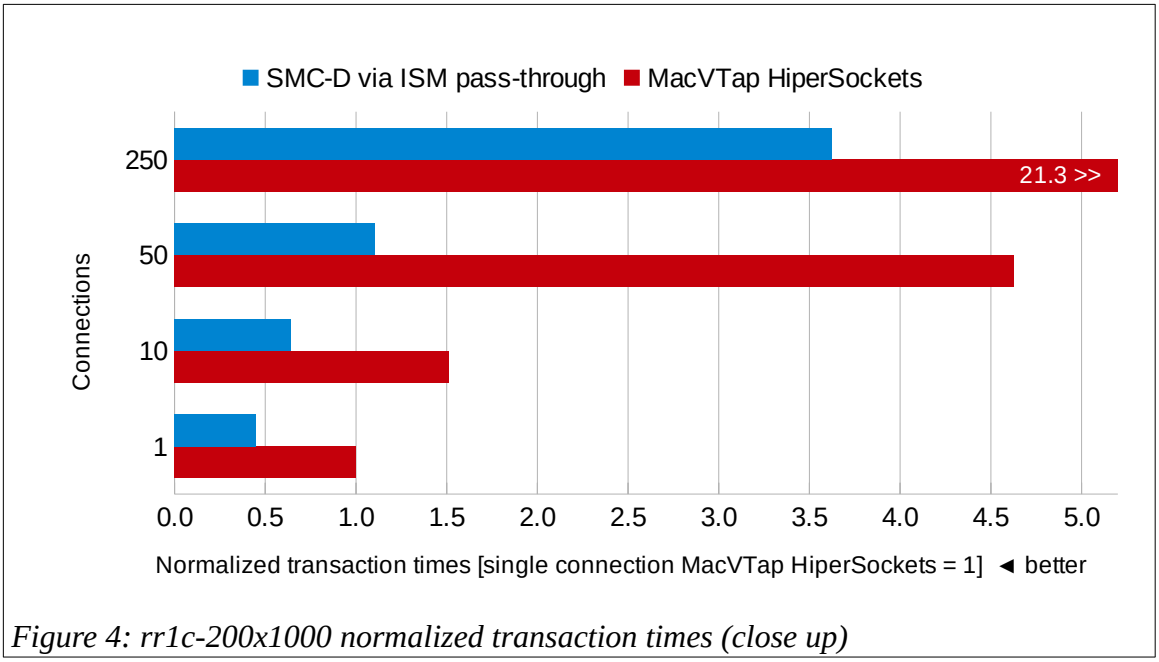

Figure 4: rr1c-200x1000 normalized transaction times (close up)

For the single-connection case the transaction times of SMC-D via ISM pass-through are less than half compared to MacVTap HiperSockets. With an increasing number of connections SMC-D becomes even more beneficial. Note that figure 4 depicts a closeup where the 250-connection case for MacVTap HiperSockets is truncated.

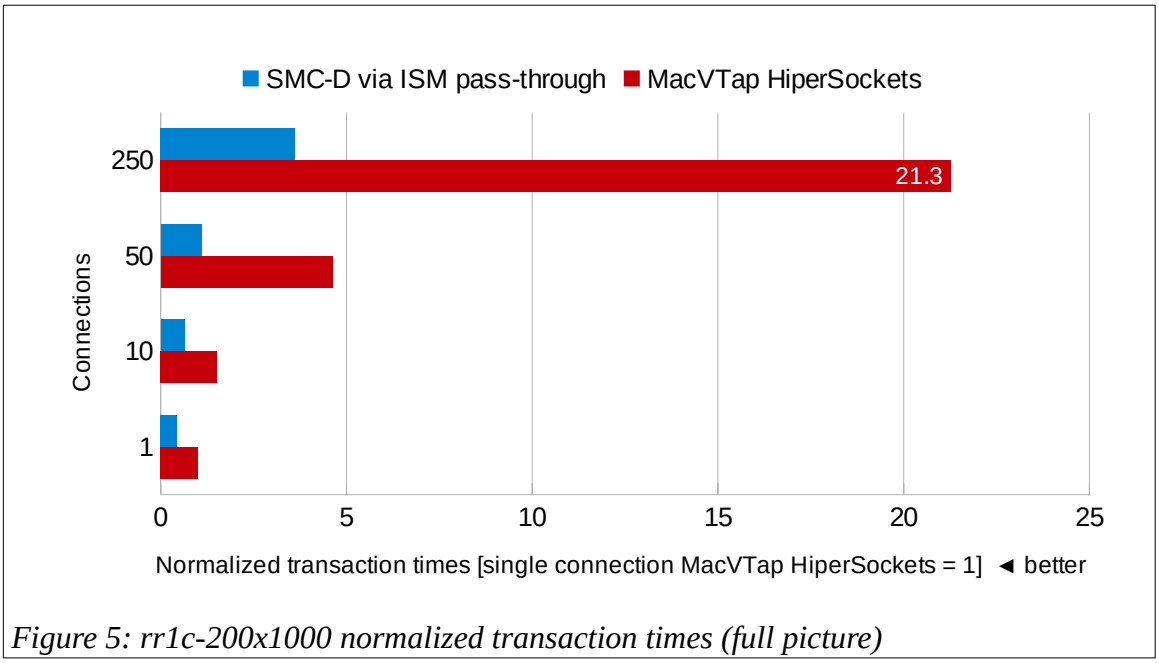

Figure 5: rr1c-200x1000 normalized transaction times (full picture)

Figure 5 shows the full picture of the same measurement and illustrates that for 250 connections SMC-D yields 83% lower transaction times than MacVTap HiperSockets.

Figure 6 belongs to the same set of measurements described above. It depicts the processor consumption of LPAR A – including KVM host and KVM guest (*uperf* server) – comparing MacVTap HiperSockets and SMC-D via ISM pass-through. Processor consumption is measured in µs/Gb and normalized to the single-connection MacVTap HiperSockets case like in figures 4 and 5.

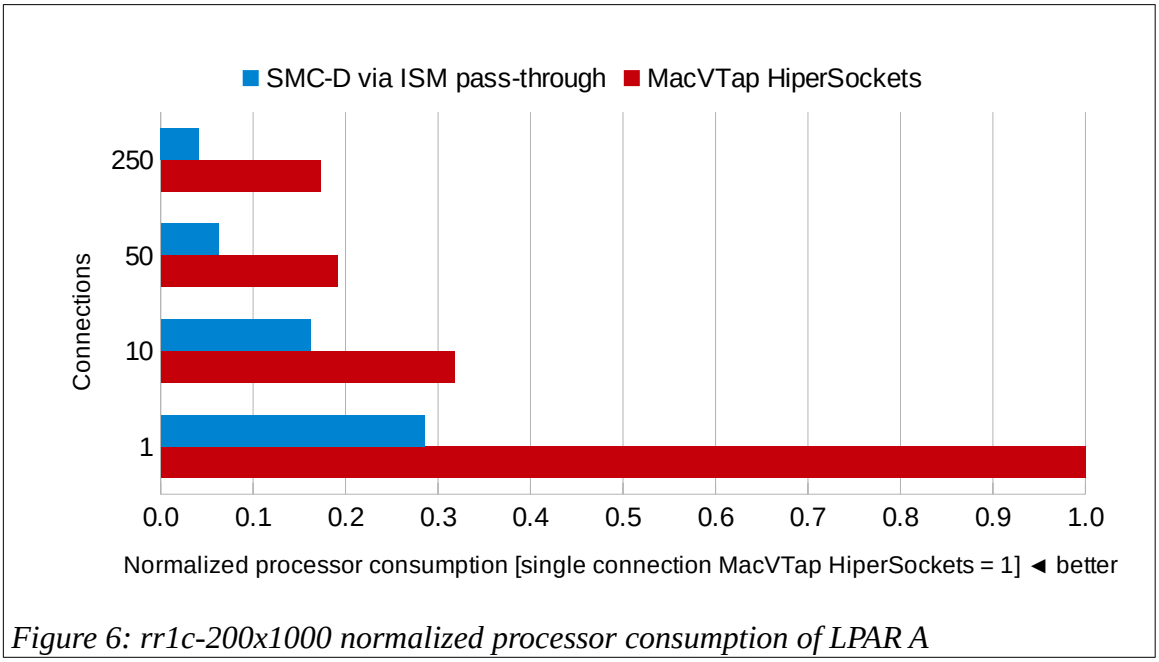

Figure 6: rr1c-200x1000 normalized processor consumption of LPAR A

The reduced transaction times for SMC-D go hand in hand with a lower processor consumption. CPU time savings range from 49% for 10 connections up to 76% for 250 connections. Single connection also yields significant savings with 71% lower transaction times.

7.2 Streaming workload

Figure 7 depicts a throughput comparison between MacVTap HiperSockets and SMC-D via ISM pass-through for the streaming read workload (str-readx30k).

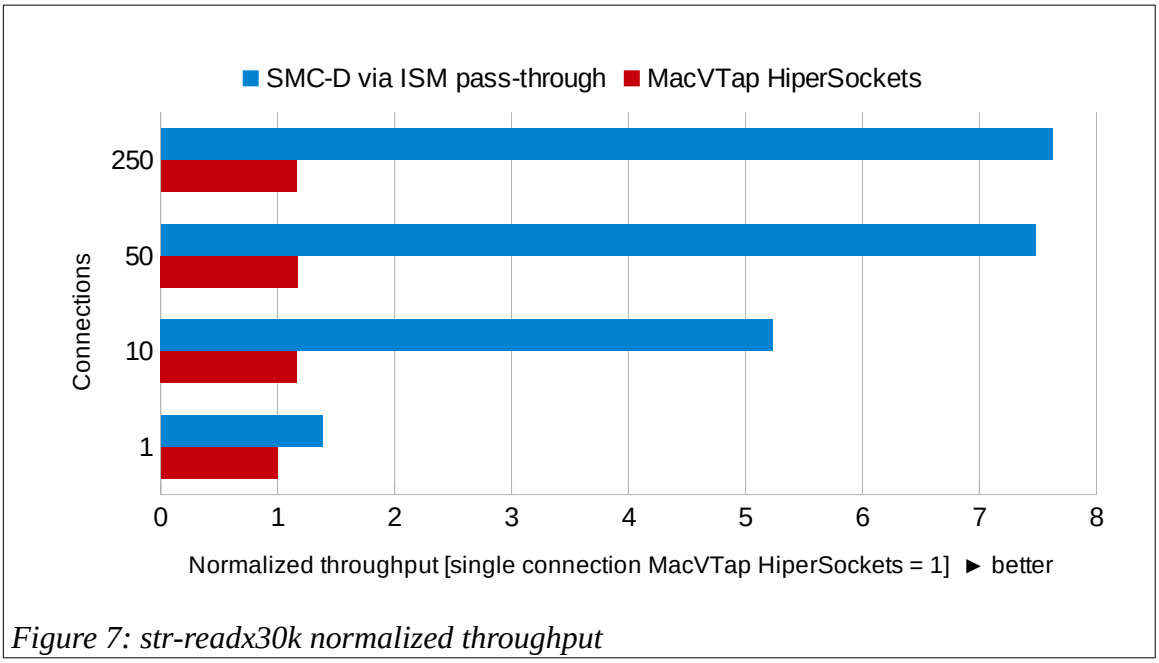

Figure 7: str-readx30k normalized throughput

As before, the values have been normalized to the single-connection MacVTap HiperSockets case. Longer bars mean higher throughput and thus are preferred.

For the single-connection case, SMC-D achieves 38% higher throughput than MacVTap HiperSockets. Furthermore, SMC-D throughput also scales well as the number of parallel connections is increased. This cannot be observed with MacVTap HiperSockets where throughput reaches saturation with as little as 10 connections.

With 50 parallel connections SMC-D outperforms MacVTap HiperSockets by a factor of nearly 6.4. At this point all available SMT-2 threads of LPAR A are almost fully utilized (in transferring data). As a consequence, with 250 connections there is only slight increase in throughput resulting in a factor of 6.6 compared to MacVTap HiperSockets. This is because the system was already heavily CPU bound with 50 connections.
Thus, an important observation gained from the tests is that SMC-D via ISM pass-through is very efficient in terms of utilizing all available SMT-2 threads, especially for large request-response and streaming workloads with multiple connections. This implies that adding more CPU cores is most likely beneficial for multi-connection workloads where an additional increase in throughput is required.

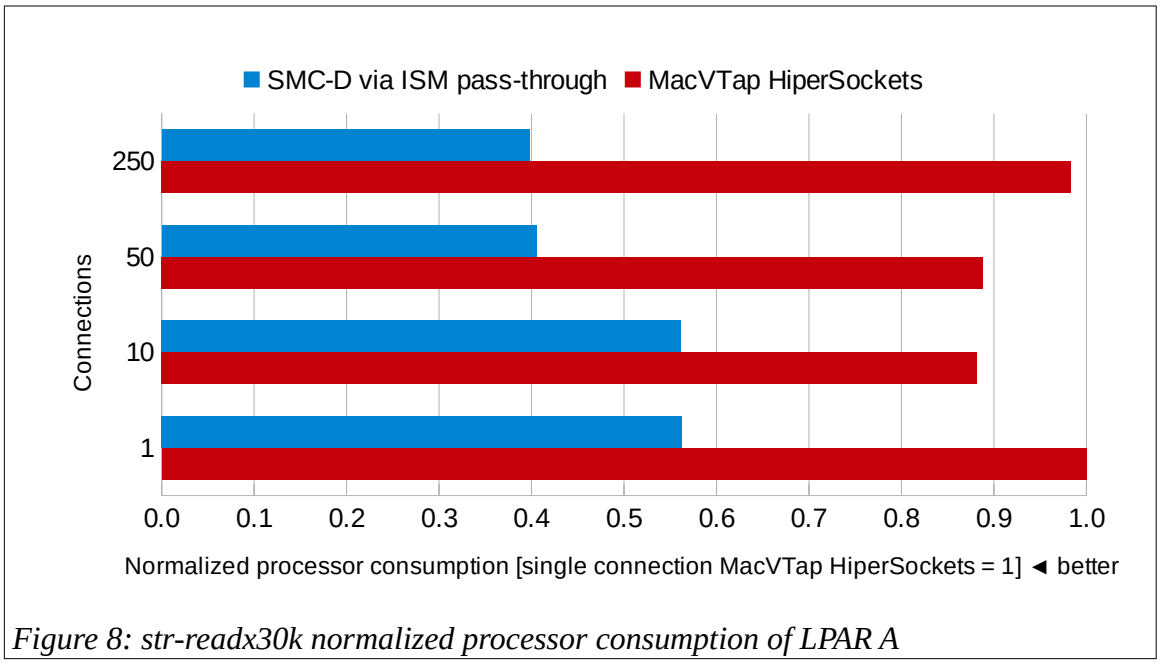

Figure 8: str-readx30k normalized processor consumption of LPAR A

Figure 8 shows a comparison of the normalized processor consumption between SMC-D via ISM pass-through and MacVTap HiperSockets of LPAR A – the KVM server – for the streaming workload.
While SMC-D already outperforms MacVTap HiperSockets in terms of throughput, the processor consumption (µs/Gb) is lower as well. CPU time savings range from 36% for 10 connections up to 60% for the 250 connections case. Single connection processor consumption savings for SMC-D via ISM pass-through amount to a value of 44%.
Furthermore, MacVTap HiperSockets show an increase in processor consumption when going from 50 to 250 connections.

7.3 SMC-D via ISM pass-through with buffer-size tuning

To investigate the influence of buffer sizes, all SMC-D measurements were repeated with larger SMC receive and transmit buffers. The results were compared with the previous measurements that used SMC buffer sizes based on default settings.

The only network tuning that has been applied so far was to increase the buffer count for HiperSockets (see chapter 6.1.2). Apart from this, no further tuning was implemented and only default distribution settings were used. In particular receive and transmit buffer sizes have been left untouched at their RHEL 8.8 defaults. The *sysctl* command can be used to display and alter various kernel parameters, including buffer sizes:

```
# sysctl net.ipv4.tcp_rmem
net.ipv4.tcp_rmem = 4096      87380 6291456
# sysctl net.ipv4.tcp_wmem
net.ipv4.tcp_wmem = 4096      16384 4194304
```

The three values represent the minimum, default and maximum sizes of the TCP receive and transmit buffers.

For SMC-D an extra set of SMC receive and transmit buffers is created. Their size is derived from the default values of the TCP receive and transmit buffers[1]. The calculation of the SMC buffer size has changed in different kernel versions. In RHEL 8.8 with kernel version 4.18.0-477.10.1.el8_8.s390x the calculation is carried out as follows: First, if the default TCP buffer size is not a power of two already, its value is rounded up to the next higher power of two. In a second step, half of this number determines the size of the SMC buffer under the condition that its minimum size is 16 KiB.
To illustrate this, the following snippet shows the SMC buffer sizes for a workload with 50 connections. The SMC buffer sizes in bold have been derived from the settings of the default TCP buffer sizes shown above:

```
# smcd -d stats

[...]

RX Stats
            8KB    16KB    32KB    64KB   128KB   256KB   512KB  >512KB
  Bufs        0       0       0      51       0       0       0       0

[...]

TX Stats
            8KB    16KB    32KB    64KB   128KB   256KB   512KB  >512KB
  Bufs        0      51       0       0       0       0       0       0
```

1 This only applies if the SMC buffer size was not set directly, e.g. via the *smc_run* command.

7.3.1 Applying SMC buffer-size tuning

The following tuning sets the default SMC receive and transmit buffer sizes to 128 KiB. According to the calculation outlined above, this requires the TCP default buffers to be set to 256 KiB. In the test environment, the following *sysctl* settings were applied to both, the KVM guest as the server and the client LPAR:

```
# sysctl net.ipv4.tcp_rmem="4096 262144 6291456"
# sysctl net.ipv4.tcp_wmem="4096 262144 4194304"
```

For persistent overrides the `/etc/sysctl.conf` configuration file can be used instead:

```
net.ipv4.tcp_rmem="4096 262144 6291456"
net.ipv4.tcp_wmem="4096 262144 4194304"
```

The snippet below shows the resulting 128 KiB SMC buffers derived from the increased 256 KiB default TCP buffer setting, again for a workload with 50 connections:

```
# smcd -d stats

[...]

RX Stats
            8KB    16KB    32KB    64KB   128KB   256KB   512KB  >512KB
  Bufs        0       0       0       0      51       0       0       0

[...]

TX Stats
            8KB    16KB    32KB    64KB   128KB   256KB   512KB  >512KB
  Bufs        0       0       0       0      51       0       0       0
```

Note that above settings are applied on a per-socket basis. Therefore setting buffer sizes too large wastes memory. Furthermore it should be mentioned that setting SMC buffer sizes via `net.ipv4.tcp_rmem[1]` and `net.ipv4.tcp_wmem[1]` also changes the default sizes of the TCP buffers. For most applications, however, this effect should be negligible.

The *sysctl* settings of the SMC buffer sizes are queried every time a new connection is established (i.e. every time a new socket is created) and adopted for the new connection. This means that changing the *sysctl* settings for the server is still possible even if it has already been started via the *smc_run* command. The modified *sysctl* settings of SMC buffer sizes then take effect for new connections from the client. However, it is not possible to change SMC buffer size for existing connections.

Another possibility to set SMC buffer sizes is to use the ‘`-r`’ and ‘`-t`’ switches of the *smc_run* command (see [5]: smc_run - Run a TCP socket program with the SMC protocol using a preloaded library).

Thus, instead of altering *sysctl* values, one could also directly set the SMC receive and transmit buffer sizes to 128 KiB as follows:

```
server# smc_run -r 128k -t 128k uperf -s
client# smc_run -r 128k -t 128k uperf -a -i 30 -m profile.xml
```

Those parameters override the global *sysctl* settings. This can be convenient if buffer sizes only need to be changed on a per-application basis. One could think of scenarios where a single workload generates “streaming-like” traffic, but the majority of applications use “highly transactional” patterns instead. Without getting too far ahead of the results, it might therefore be appropriate to increase the SMC buffers for a streaming application via the ‘`-r`’ and ‘`-t`’ switches of the *smc_run* command and let transactional workloads use the smaller default buffers derived from the system-wide *sysctl* settings.

Note:
Kernel version 6.5[2] introduced a set of SMC-specific *sysctl* tunables called *net.smc.rmem* and *net.smc.wmem*. They allow SMC buffer sizes to be set independently from the TCP buffers.

7.3.2 Impact of buffer-size tuning on the request-response workload

In the test setup, tuning SMC buffer sizes has a slightly negative impact on transaction times for the single-connection SMC-D request-response workload of about 4%. However, SMC-D request-response transaction times for 10 and more connections remain unaffected by buffer-size tuning. Furthermore, tuning SMC buffer sizes does not have an impact on processor-consumption for the request-response workload.

7.3.3 Impact of buffer-size tuning on the streaming workload

Figure 9 shows the impact of tuning SMC buffer sizes on throughput for the streaming workload. The SMC-D measurements with default-sized buffers (as described in chapter 7.2) serve as reference. Throughput numbers are normalized to the single-connection case with default SMC buffer size.

2 The commit id of this feature is 833bac7e.

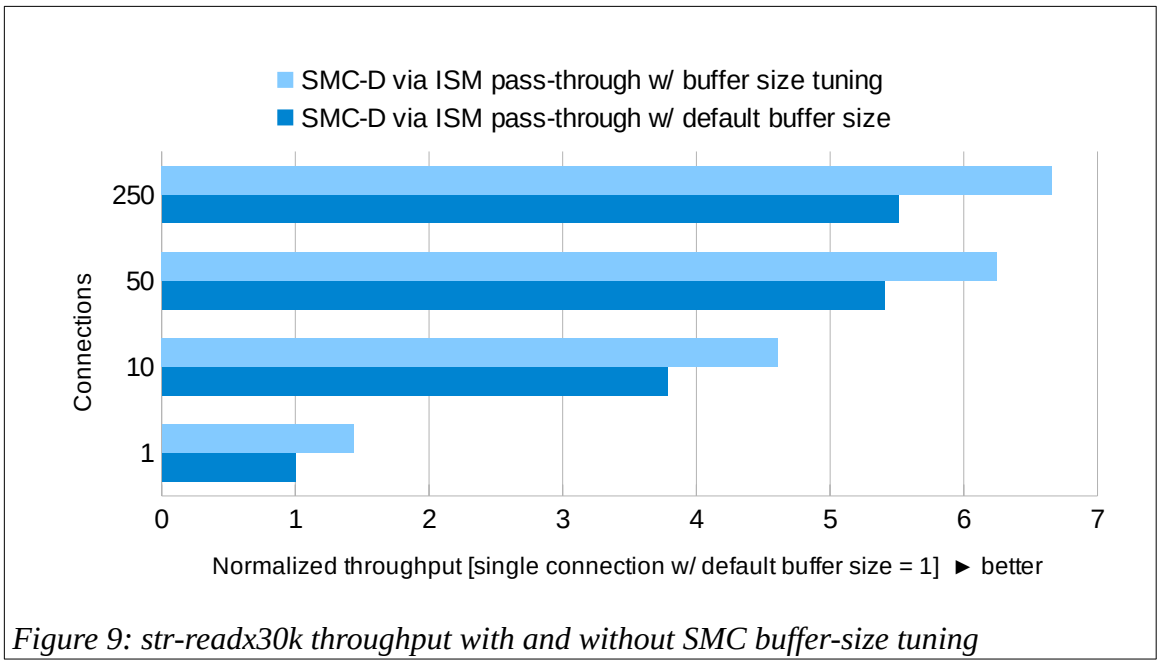

Figure 9: str-readx30k throughput with and without SMC buffer-size tuning

The results (see figure 9) show that tuning buffer sizes significantly helps to improve throughput rates, especially in the single-connection case with nearly 44% improvement. The multi-connection cases still yield throughput improvements in a range from 16% to 22%.
Similar to previous measurements with default-sized buffers, LPAR A is almost CPU bound again when using 50 or more parallel connections. Thus, adding more processor cores might help to further increase throughput for scenarios with a high number of parallel connections.

Finally, figure 10 illustrates how buffer-size tuning contributes to saving processor consumption (µs/Gb) for the streaming workload.

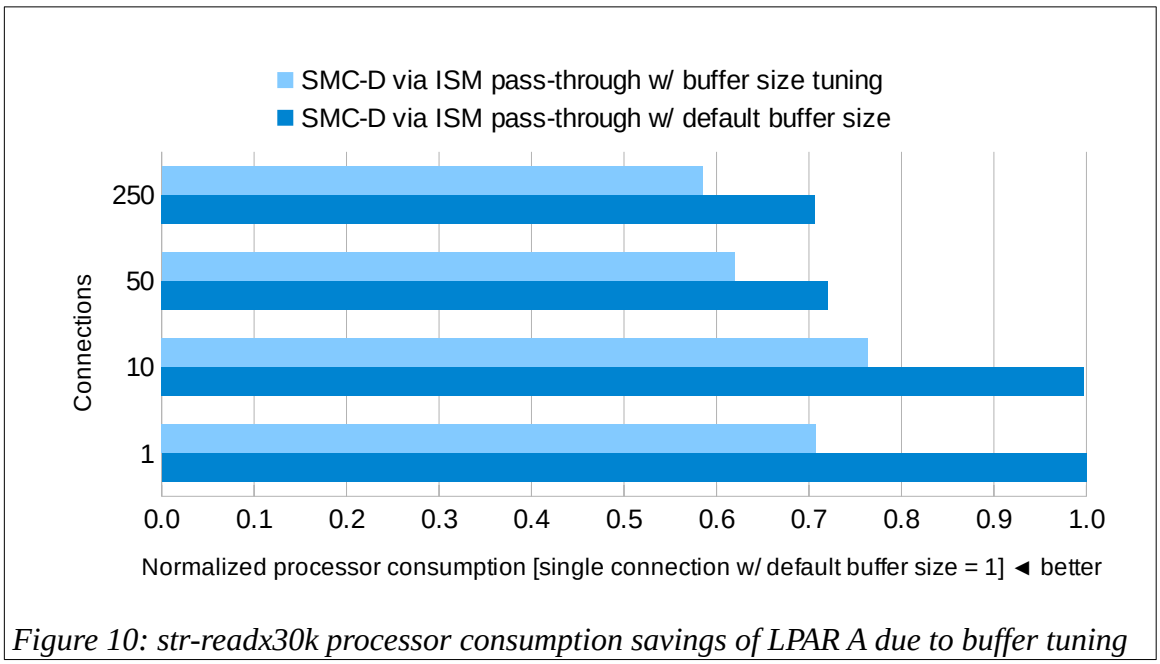

Figure 10: str-readx30k processor consumption savings of LPAR A due to buffer tuning

The savings range from 14% to 29%, again with the largest savings for the single-connection case.

8 Takeaways

This study compares two possible networking options for guest-to-LPAR communication within a CPC: MacVTap HiperSockets and SMC-D via ISM pass-through.

SMC-D via ISM pass-through outperforms MacVTap HiperSockets for highly transactional/latency-critical request-response traffic patterns with up to 83% reduced latency. Additionally, SMC-D can increase throughput of streaming workloads by up to 6.6 times. These performance benefits go hand in hand with significant processor consumption savings in the high double-digit range.

The streaming performance of SMC-D can be further enhanced by increasing SMC buffer sizes. With the selected tuning of 128 KiB for SMC transmit and receive buffers, throughput was increased by up to 44% and processor consumption reduced by up to 29% – both compared to performance metrics using default sized buffers. It should be noted that SMC-D streaming workloads could benefit from even larger SMC buffers.

SMC-Dv2 is available for IBM z15 and z16 with the ISM feature enabled.
PCI device pass-through for KVM guests is supported with the following (or later) Linux on IBM Z distributions: Red Hat Enterprise Linux 8.8 and 9.2, SUSE Linux Enterprise Server 15 SP5 and Ubuntu Server 23.04.

9 Appendix

9.1 Output of *lsqeth* command for HiperSockets

The following output of the *lsqeth* command shows the characteristics of a HiperSockets adapter in layer 2 mode with an increased buffer count of 128 as used in the test environment:

```
kvmhost# lsqeth
Device name                     : hs32k_0
-------------------------------------------------------------------------------
        card_type               : HiperSockets
        cdev0                   : 0.0.8200
        cdev1                   : 0.0.8201
        cdev2                   : 0.0.8202
        chpid                   : FC
        online                  : 1
        portname                : no portname required
        portno                  : 0
        state                   : UP (LAN ONLINE)
        priority_queueing       : disabled
        buffer_count            : 128
        layer2                  : 1
        isolation               : none
        bridge_role             : none
        bridge_state            : inactive
        bridge_hostnotify       : 0
        bridge_reflect_promisc  : none
        vnicc/bridge_invisible  : 0
        vnicc/flooding          : 0
        vnicc/learning          : 0
        vnicc/learning_timeout  : 600
        vnicc/mcast_flooding    : 0
        vnicc/rx_bcast          : 1
        vnicc/takeover_learning : 0
        vnicc/takeover_setvmac  : 0
```

9.2 `profile.xml` for a medium-size request-response workload

The following *uperf* `profile.xml` defines a medium-size request-response workload with 50 parallel connections and a runtime of 300 seconds:

```
<?xml version="1.0"?>
<profile name="TCP_RR">
  <group nprocs="50">
  <!-- if one wants to run threads instead of processes -->
  <!--group nthreads="50"-->
    <transaction iterations="1">
      <flowop type="connect" options="remotehost=10.x.y.1
       protocol=tcp tcp_nodelay" />
    </transaction>
    <transaction duration="300">
      <flowop type="write" options="size=200"/>
      <flowop type="read" options="size=1000"/>
    </transaction>
    <transaction iterations="1">
      <flowop type="disconnect" />
    </transaction>
  </group>
</profile>
```

9.3 SMC-D connection statistics for a medium-size request-response workload

The SMC-D connection statistics shown below refer to a medium-size request-response workload with 50 parallel connections:

```
client# smcd -d stats
SMC-D Connections Summary
  Total connections handled            51
  SMC connections                      51 (client 51, server 0)
    v1                                  0
    v2                                 51
  Handshake errors                      0 (client 0, server 0)
  Avg requests per SMC conn      14046725.0
  TCP fallback                          0 (client 0, server 0)

RX Stats
  Data transmitted (Bytes)   358191451596 (358.2G)
  Total requests                358191491
  Buffer full                           0 (0.00%)
  Buffer downgrades                     0
  Buffer reuses                        51
            8KB    16KB    32KB    64KB   128KB   256KB   512KB  >512KB
  Bufs        0       0       0      51       0       0       0       0
  Reqs   358.2M       1       0       0       0       0       0       0

TX Stats
  Data transmitted (Bytes)    71638302336 (71.64G)
  Total requests                358191501
  Buffer full                           0 (0.00%)
  Buffer full (remote)                  0 (0.00%)
  Buffer too small                      0 (0.00%)
  Buffer too small (remote)             0 (0.00%)
  Buffer downgrades                     0
  Buffer reuses                        51
            8KB    16KB    32KB    64KB   128KB   256KB   512KB  >512KB
  Bufs        0      51       0       0       0       0       0       0
  Reqs   358.2M       0       0       0       0       0       0       0

Extras
  Special socket calls                 50
    cork                                0
    nodelay                            50
    sendpage                            0
    splice                              0
    urgent data                         0
```

10 Bibliography

1: IBM Washington Systems Center, Shared Memory Communications – Direct Memory Access (SMC-D) Frequently Asked Questions, 2023, https://www.ibm.com/support/pages/node/7004951

2: Randall Kunkel, Jerry Stevens, IBM Shared Memory Communications: Version 2, Third Edition, 2023, https://www.ibm.com/support/pages/ibm-shared-memory-communications-version-2

3: Nils Hoppmann, Exploring the performance of network adapters for Linux on IBM Z, 2021, https://www.ibm.com/docs/en/linuxonibm/pdf/z_performance_of_network_adapters_Linux_on_IBM_Z.pdf

4: Mike Ebbers, Micky Reichenberg, Alexandra Winter, IBM HiperSockets Implementation Guide (SG24-6816-02), 2014, https://www.redbooks.ibm.com/redbooks/pdfs/sg246816.pdf

5: Device Drivers, Features, and Commands on Red Hat Enterprise Linux 8.6 (SC34-7715-06), 2022, https://www.ibm.com/docs/en/linux-on-systems?topic=commands-red-hat-enterprise-linux-86

6: KVM Virtual Server Management (SC34-2752-08), 2022, https://www.ibm.com/docs/en/linux-on-systems?topic=management-november-2022

7: Dr. Juergen Doelle, KVM Network Performance Best Practices and Tuning Recommendations, 2020, https://public.dhe.ibm.com/software/dw/linux390/perf/KVM_Network_Performance-Best_Practices_and_Tuning.pdf

8: Stefan Raspl, libvirt v4.10 released, providing PCI passthrough support, 2019, https://kvmonz.blogspot.com/2019/01/libvirt-v410-released-providing-pci.html

9: Mark A. Peloquin, Dr. Juergen Doelle, KVM Network Performance - Best Practices and Tuning Recommendations, 2018, https://www.ibm.com/docs/en/linux-on-systems?topic=kvm-network-performance-best-practices-tuning-recommendations

11 Notices and disclaimer

U.S. Government Users Restricted Rights – use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts. In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply.

Any statements regarding IBM’s future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in a controlled, isolated environment. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer’s responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer’s business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM’s products.

IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: https://www.ibm.com/legal/copyright-trademark.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right. Refer to https://www.ibm.com/legal for further legal information.