KVM on IBM Z

KVM Network Performance - Best Practices and Tuning Recommendations

KVM on IBM Z

KVM Network Performance - Best Practices and Tuning Recommendations

Before using this information and the product it supports, read the information in “Notices” on page 61.

Edition notices

Contents

Figures

About this publication

This paper explores different system configurations (running KVM guests) and different networking configuration choices, as well as tuning recommendations for the KVM host and KVM guest environments to achieve greater network performance on the IBM Z platforms.

Authors

Mark A. Peloquin, Juergen Doelle

Remarks

The web-links referred in this paper are up-to-date as of October, 2016.

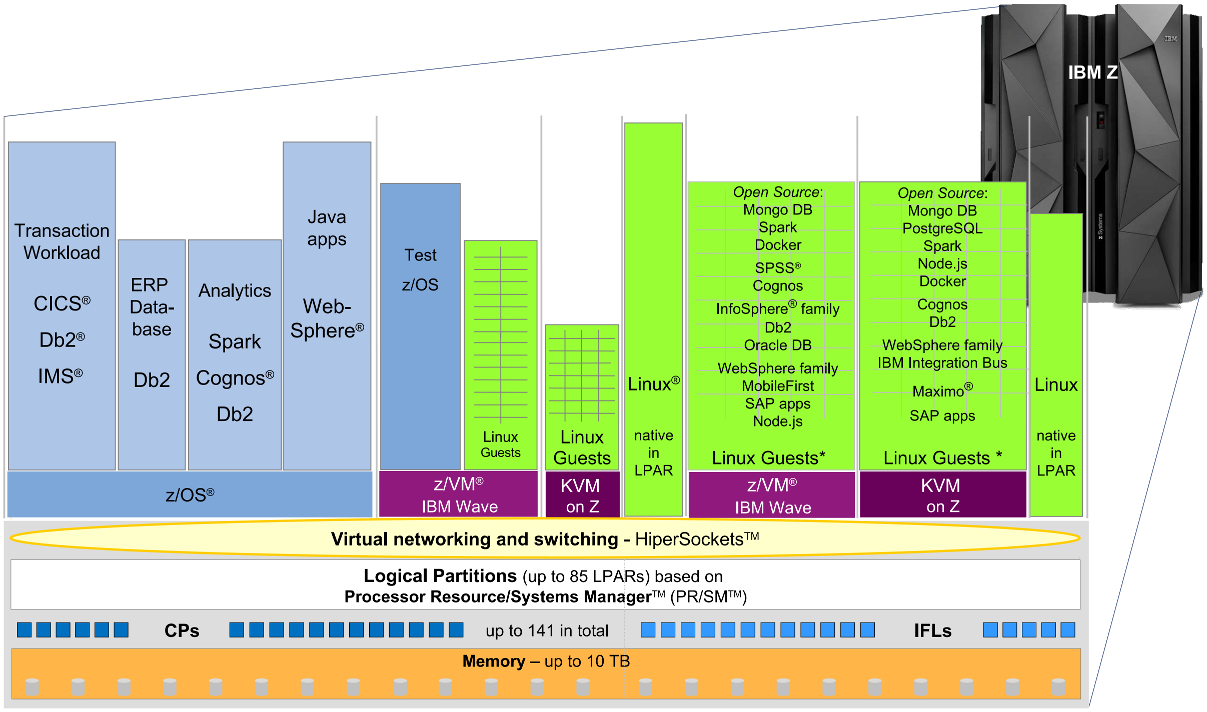
Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Measurements have been made on development-level code and there is no guarantee that these measurements will be the same on generally available releases. Actual results may vary.

Chapter 1. Overview

The IBM Z platforms are the latest generation in the IBM® family of mainframe offerings. The IBM mainframes have a long legacy from being first produced over 50 years ago. The system technology has continued to expand and evolve enabling the IBM mainframe releases to continue to offer capabilities and features enterprise customers require.

The IBM Z family maintains full backward compatibility. This means that current systems are the direct, lineal descendants of System/360 announced in 1964, and System/370 from the 1970s. Many applications written for these systems can still run unmodified on the newestIBM Z system over five decades later![1]

Over several decades, as more and more software stacks and operating system choices have been added to the IBM Z platforms software catalog and while maintaining full backward compatibility, the IBM mainframe has become a "virtual Swiss Army knife" in terms of all the things it can do.



* some workload examples

Figure 1. High-level Overview of the IBM Z Mainframe

Figure 1 provides a high-level picture of the IBM Z integrated virtualization capabilities.

- PR/SM™ allows the system resources (CPUs, memory, IO adapters) to be dedicated or shared across up to 60 logical partitions.

[1] IBM Z on Wikipedia

- Each of the logical partitions can be installed with a variety of target operating systems.

The z/OS® and z/VM® operating systems have been available for a long time. For a number of years, the choice of running the Linux operating system has been supported. Now the support for the hypervisor KVM has been included to provide open-source virtualization capabilities within a logical partition on IBM Z and IBM LinuxONE™ platforms.

Using the combination of KVM virtualization and IBM Z and IBM LinuxONE, you have the performance and flexibility to address the requirements of multiple, differing Linux workloads. KVM's open source virtualization on IBM Z and LinuxONE allow businesses to reduce costs by deploying fewer systems to run more workloads, sharing resources, and improving service levels to meet demand.[2]

For more information about the LinuxONE and KVM running on IBM Z, visit IBM developerWorks® (http://www.ibm.com/developerworks/linux/kvmforz/index.html).

[2] Linux on KVM

Chapter 2. Introduction

This paper documents the analysis and findings of the Network Performance analysis that was conducted using Kernel-based Virtual Machines (*KVM*) running on Linux developed for IBM Z.

KVM is a virtualization infrastructure that enables the Linux kernel to become a hypervisor with the ability to run a separate and distinct operating system in a virtual machine. IBM Z platforms now support the use of KVM to create and run multiple virtual machines or guests in each LPAR.

Note: The availability of the features mentioned in this paper might depend on the KVM release.

Chapter 3. Test environment

The network performance analysis used two basic IBM Z configurations.

Each configuration uses a single or a pair of logical partitions (LPARs) on IBM Z.

> A logical partition is a subset of a computer's hardware resources, *virtualized* as a separate computer. In effect, a physical machine can be partitioned into multiple logical partitions, each hosting a separate *operating system*.[3]

Up to two LPARs on an IBM z13® system are used. The configuration for each LPAR is identical and contains the following characteristics:

- 32 IFL (Integrated Facilities for Linux) processors
- 128 GB memory
- One 1GbE network connection configured from an Open System Adapter (OSA) 4 card. This connection was used on a public subnet to provide administrative access to each LPAR.
- Two 10GbE network connections, each configured from separate OSA 4 adapters. Both OSA adapters were configured in both LPARs. Note that having two OSAs allows testing shared OSA as well as cross-OSA, connecting one OSA to the other via the switch. The 10GbE links were connected to a dedicated network switch to create an isolated network environment. The interfaces were configured to use the private subnet and used exclusively for network performance measurement testing.
- Eight Fibre Connection (FICON®) storage adapters providing multipath SAN access to Extended Count Key Data (ECKD™) Direct Access Storage Devices (DASD) on a DS8000® storage controller. Each LPAR uses as single DASD disk as the boot and install device.
- Eight Fiber Channel Protocol (FCP) adapters providing multipath SAN access to SCSI devices on the DS8K storage controller. A SCSI device is used as the boot/install device used by each KVM virtual machines / guests.

Note: Both 10GbE interfaces were configured in all KVM guests for test configuration convenience.

However, for all network configurations and all test measurements, only a single 10GbE was actually used at any point in time. Having multiple 10GbE interfaces configured enabled tests using KVM guests on separate KVM hosts to use a separate 10GbE interface (on a separate OSA adapter) for the KVM guests running on each KVM host. Using separate OSA adapters ensures network traffic flows out of the KVM host to the switch and back to the destination system (see Figure 7 on page 15), avoiding any KVM host or OSA adapter shortcuts or optimizations, to capture the performance and behavior seen when communicating with a system external from the KVM host.

[3] logical partition: Wikipedia

Single KVM host

The first configuration uses a single LPAR installed with Linux containing KVM support. KVM virtual machines or guests are configured and installed with an operating system.

This configuration tests the networking performance available in a single KVM host environment using various network modes to connect and direct network traffic from one KVM guest to another KVM guest. In this configuration, all guest to guest network traffic is typically handled internal to the KVM host and does ***not*** flow externally over the physical network adapter to a remote system or LPAR. This configuration is designed to ***stress*** the traffic flowing from one KVM guest to another KVM guest on the same host.

Since the guests on the same host are connected via software components (virtual drivers, bridges or switches) the maximum achievable network speed is not limited by the speed of physical network hardware. This configuration allows cross-guest network traffic to (sometimes greatly) exceed the speed of the physical adapter hardware. Instead of being limited by the physical network fabric speed, which has the potential to be the slowest element in the network path, other factors such as the available (host or guest) CPU resources or the system's memory bandwidth could become the limitation. By removing the limitations imposed by physical network fabric, virtual networking in a single KVM host configuration can produce greater ***stress*** on the network stack and components.

Figure 2 illustrates the block level diagram of the single LPAR configurations.

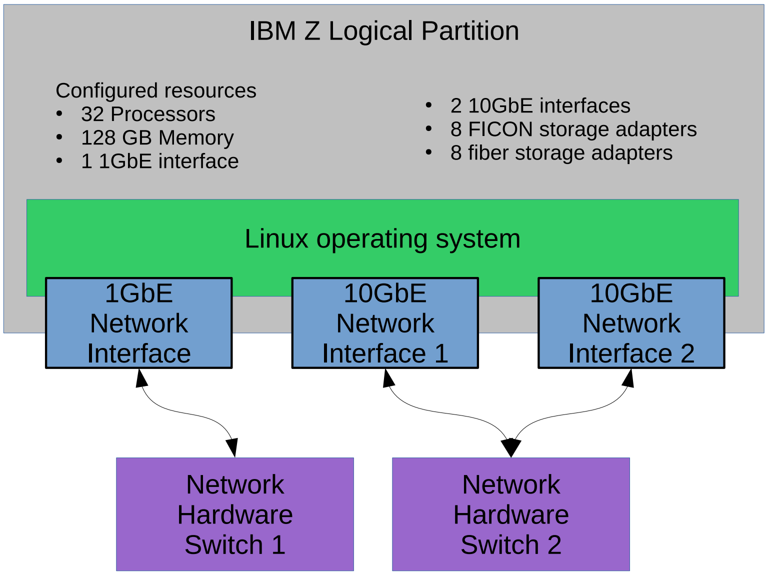


Figure 2. Single LPAR - KVM Host Configuration

Multiple KVM hosts

The second configuration uses two LPARs both installed with Linux and KVM support.

As in the single LPAR configuration, both LPARs are KVM hosts and have KVM guests configured and installed. In this configuration, the KVM guests in the first LPAR are used to connect and interact with KVM guests from the second LPAR. This configuration is designed to use and test the physical network pathway that provide communication in and out of the LPARs as well as demonstrating the ability to drive the physical links to full utilization.

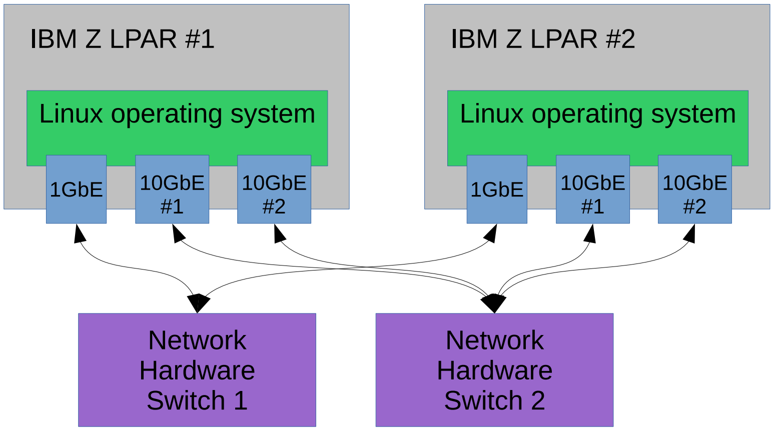


Figure 3. Two LPAR / KVM Host Configuration

KVM guest configuration

Each LPAR was configured to support sixteen KVM guests.

The configuration for all KVM guests was identical and contained the following details:

- 4 virtual processors
- 512 MB of memory
- 1 SCSI device used for installation and boot of Linux.
- 2 network interfaces. Each interfaces is connected to a separate 10GbE interface in the KVM host LPAR.

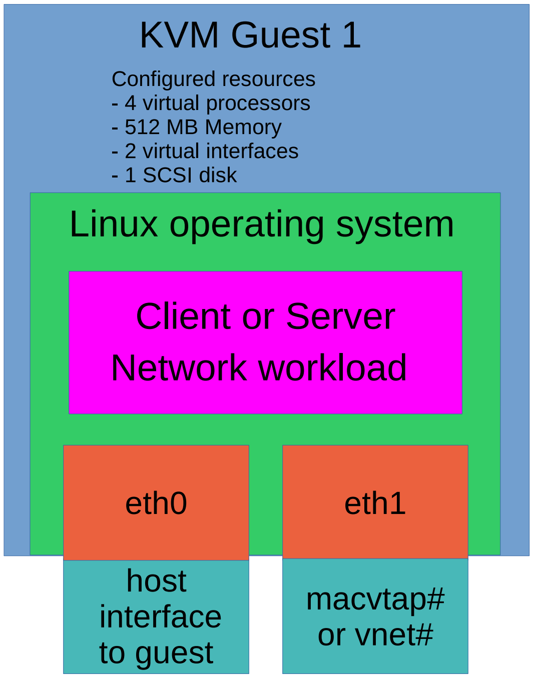


Figure 4. KVM Guest configuration

KVM host network configuration

KVM on Z supports a variety of host OS network configuration methods that are available for KVM guests to use.

This paper evaluates three network configuration choices that provide connectivity to the KVM guests:

1. KVM guests using Linux bridge mode
2. KVM guests using OpenvSwitch as substitute for a Linux bridge
3. KVM guests using the MacVTap driver.

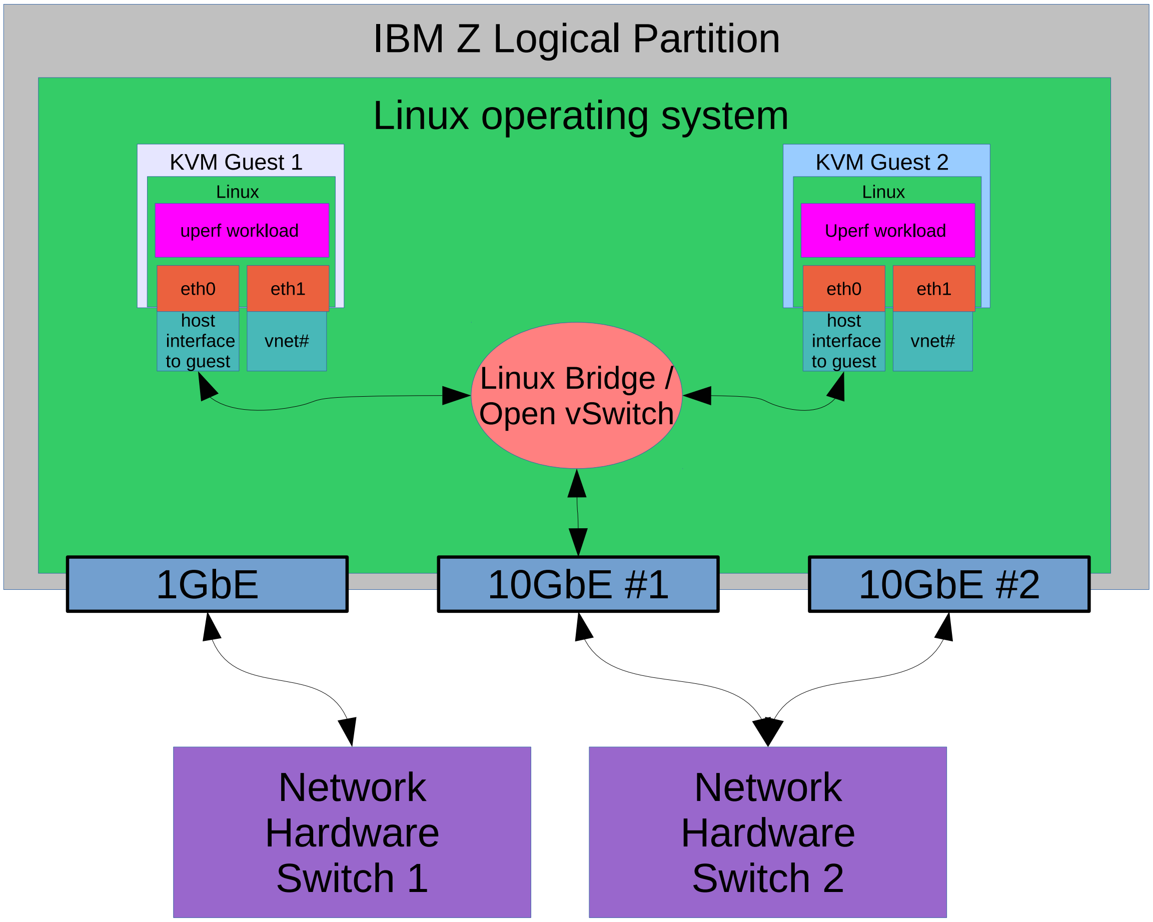


Figure 5. KVM guest configuration using a software bridge

Note: The KVM guests shown above are a smaller version of those shown in Figure 4 on page 8.

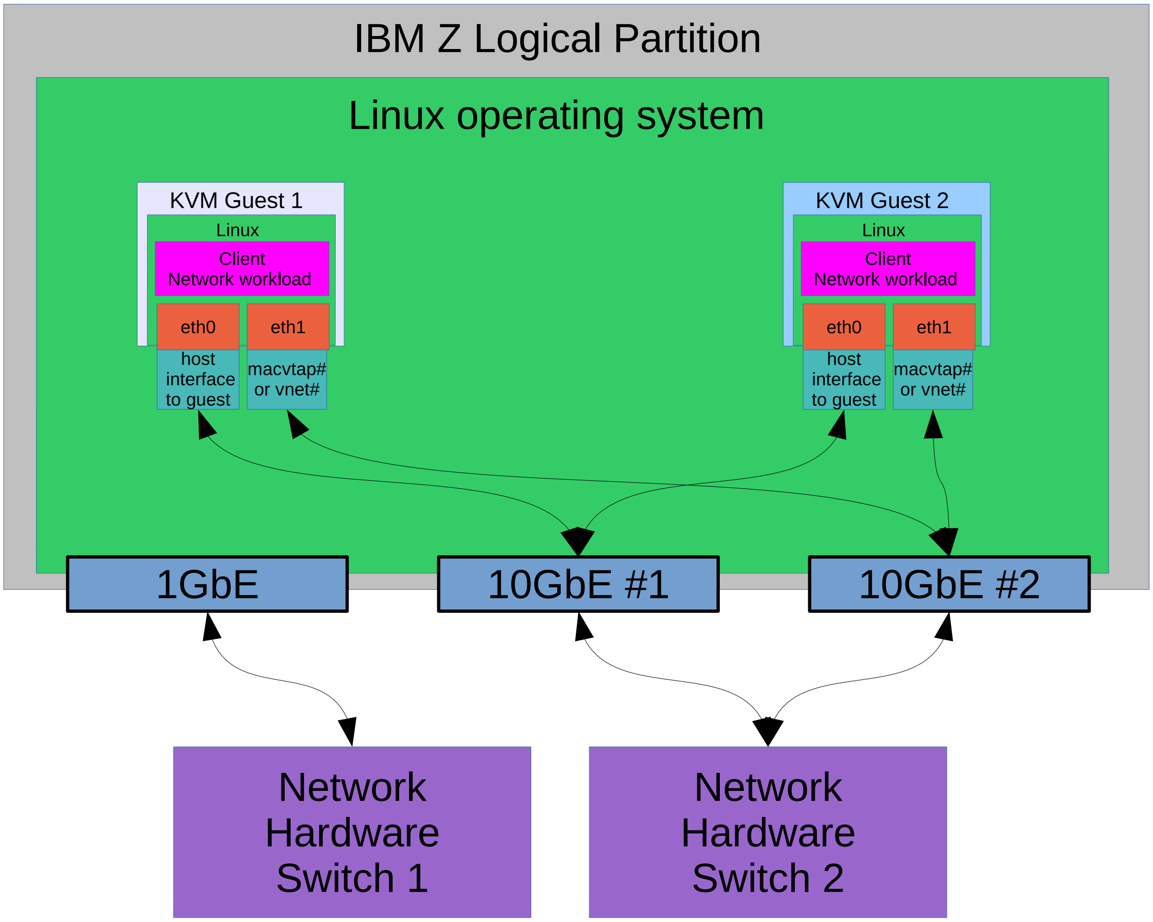


Figure 6. KVM guests using a MacVTap Network Configuration

Operating system versions

The goal of this paper is to help customers (administrators and users) using the KVM hypervisor running on a IBM Z platform to more efficiently set up, configure and tune their configurations to obtain better network performance.

The results presented in this paper were collected from an IBM internal development driver running in the KVM hosts which may contain features and enhancements not yet available in current KVM releases, but which are planned for inclusion in upcoming releases. The vast majority of recommendations and tunings are relevant to all KVM releases.

Chapter 4. Network workload

The network workload was generated by using the network performance tool `uperf`.

Description of `uperf`

From www.uperf.org: “`uperf` is a network performance tool that supports modelling and replay of various networking patterns ... `uperf` represents the next generation benchmarking tools (like filebench) where instead of running a fixed benchmark or workload, a description (or model) of the workload is provided and the tool generates the load according to the model. By distilling the benchmark or workload into a *model*, you can now do various things like change the scale of the workload, change different parameters, change protocols, and so on, and analyze the effect of these changes on your model. You can also study the effect of interleaving CPU activity, or think times or the use of SSL instead of TCP among many other things.”

`uperf` profiles

`uperf`[4] uses a profile in XML format to define the desired characteristics of the workload that will run. An example of a profile describing a test that sends a 1-byte request, and receives a 1-byte response over a single connection:

```
<?xml version="1.0"?>
<profile name="TCP_RR">
  <group nprocs="1">
    <transaction iterations="1">
      <flowop type="accept" options="remotehost=10.12.37.2 protocol=tcp tcp_nodelay"/>
    </transaction>
    <transaction duration="300">
      <flowop type="write" options="size=1"/>
      <flowop type="read" options="size=1"/>
    </transaction>
    <transaction iterations="1">
      <flowop type="disconnect" />
    </transaction>
   </group>
</profile>
```

For more details about all the fields available in the `uperf` profile, refer to the www.uperf.org Web page.

As described above, `uperf` supports a wide variety of options and is highly configurable permitting the ability to model or simulate practically any network behavior desired.

Workload Configurations

The test methodology used includes workloads that have been divided into two high level categories, transactional and streaming workloads.

[4] uperf - A network performance tool

Workload categories

A transactional workload is comprised of two parts, a (send) request followed by (receiving) a response. This Request-and-Response (RR) pattern is typical of what is seen by web servers as users interact with websites using web browsers. The payload sizes for these RR patterns are relatively small.

- Requests are typically in the range from a few bytes to simulate mouse-clicks up to a few hundred bytes to represent larger URLs or form data entered and sent by a user.
- Responses are typically in tens of kilobytes (KB) to deliver web pages and up to a few megabytes (MB) to deliver the embedded images typically associated with most web content.
- The ratio of RR (send/receive) payload sizes are typically in the 1:100 to 1:1,000 range. With ratios in this range, the workload is considered bi-directional which defines the meaning of transactional.

Streaming workloads (STR) tend to simulate the load characteristics that many Enterprise or SMB servers experience when supporting operations such as backup/restore, large file transfers and other content delivery services. Streaming workloads, although comprised of a Request-and-Response, are considered uni-directional because the Request-and-Response ratio can be well over 1:1,000,000 or higher. A small request can trigger responses that are many gigabytes or more in size.

Workload types

Each workload category, Request-and-Response (RR) and Streaming (STR), is further divided in to separately-defined workload types that each possess discrete characteristics.

RR testing will be split into three workload types:

1. Small packet / high transactional workloads, exclusively 1 byte requests and 1 byte responses
2. Nominal payload size transactional workloads, 200 byte requests and 1000 byte responses
3. Large payload transactional workloads, 1000 byte requests and 30KB responses

STR testing will be split into two workload types:

1. Streaming reads, 30 KB payloads
2. Streaming writes, 30 KB payloads

The intention is that these workload types should map reasonably closely to the specific characteristics of user workloads.

Simulating Users

In addition to the 5 basic workload types (3 RR and 2 STR), its also necessary to simulate the effects of users. This means having to generate the load of a single user and scale this up to reflect the load of many users. In `uperf` this can be achieved by running each workload one or more times concurrently. `uperf` has a several ways to multiply workload concurrency. The method used to specify concurrency (used for the test in this paper) is to use an optional parameter in the **group** statement.

- The parameter `nprocs=` specifies the number of concurrent processes to create.

- Each process executes the transactions defined in the group.
- If all the defined transactions are considered to represent the activity of a single user, then the nprocs value specified effectively simulates the number of users you choose.

Workload tests

For each of the 5 workload types (3 RR and 3 STR) described in the previous topic "Workload types", 4 discrete levels of concurrency were chosen to simulate increasing levels of load from additional users (1, 10, 50, 250 users) resulting in a total of 20 different tests.

Later in this paper (starting with Figure 10 on page 34), various graphs showing performance data are included. Each graph lists each of the 20 different tests on the X-axis. The names for each of the tests have the form:

For transaction tests:

```
{category}{1c}-{requestsize}x{responsesize}--{users}
```

Where {category} is RR for the transaction Request-and-Response tests, {1c} stands for using just 1 uperf client for each uperf server, {requestsize} is the number of bytes sent by the client to the server, {responsesize} is the number of bytes in the response by the server to the client and {users} represents the total number of concurrent users (or processes) generating the overall load.

For example: **rr1c-200x1000--10** describes a Request-and-Response test sending a 200 byte request and receiving a 1000 byte response being generated each by 10 concurent users.

For streaming tests:

```
{category}-{read|write}x{payloadsize}--{users}
```

Where {category} is STR for Streaming tests, {read|write} denotes which direction relative to the client data flows, {payloadsize} is the number of bytes in each datagram and {users} represents the total number of concurrent users (or processes) generating the overall load.

For example: **str-readx30k--50** describes a streaming test read of 30KB datagrams being generated each by 50 concurrent users.

uperf pairs

The uperf implementation uses a Master / Slave model. The Master and Slave relationship can also be thought of as a Client and Server. When uperf runs the Client (Master) initiates communication with the Server (Slave). The Client invocation includes a parameter to the test definition profile. After connecting, the Client sends the Server a version of the test definition.

Since uperf requires both roles of client and server, we chose to assign each role to a separate KVM guest. Each uperf client has a unique uperf server to use. We named the association of a KVM guest client to its KVM guest server counterpart a "uperf pair". A uperf pair forms the smallest aggregation of resources that defines the building block used to scale our testing configurations. Testing starts with a single uperf pair and then adds more pairs to increase and scale the load. Each step in the scale doubles the load of the previous step. The steps used 1, 2, 4 and 8 `uperf` pairs. All `uperf` pairs will run the exact same workloads and are expected to perform and behave similarly.

In addition, recall from the previous topic “Workload tests”, for each step in the number of `uperf` pairs tested, each of the 5 workload types (3 RR and 2 STR) run. Each workload type is run with 4 different levels of concurrency 1, 10, 50 and 250, where each value of concurrency equates to a single user.

So as the number of `uperf` pairs increase, so does the load of the KVM host(s), so does the number of KVM guests and so does the number of simulated users. With 8 `uperf` pairs, each using two KVM guests, one for the client and one for the server, using a total of 16 KVM guests, the define workload types will each simulate up to 2000 concurrent users.

The association of which KVM guests are used to compose a `uperf` pair differs depending on the KVM host configuration. In the single KVM host configuration, the `uperf` pairs are assembled from KVM guests running on that same KVM host.

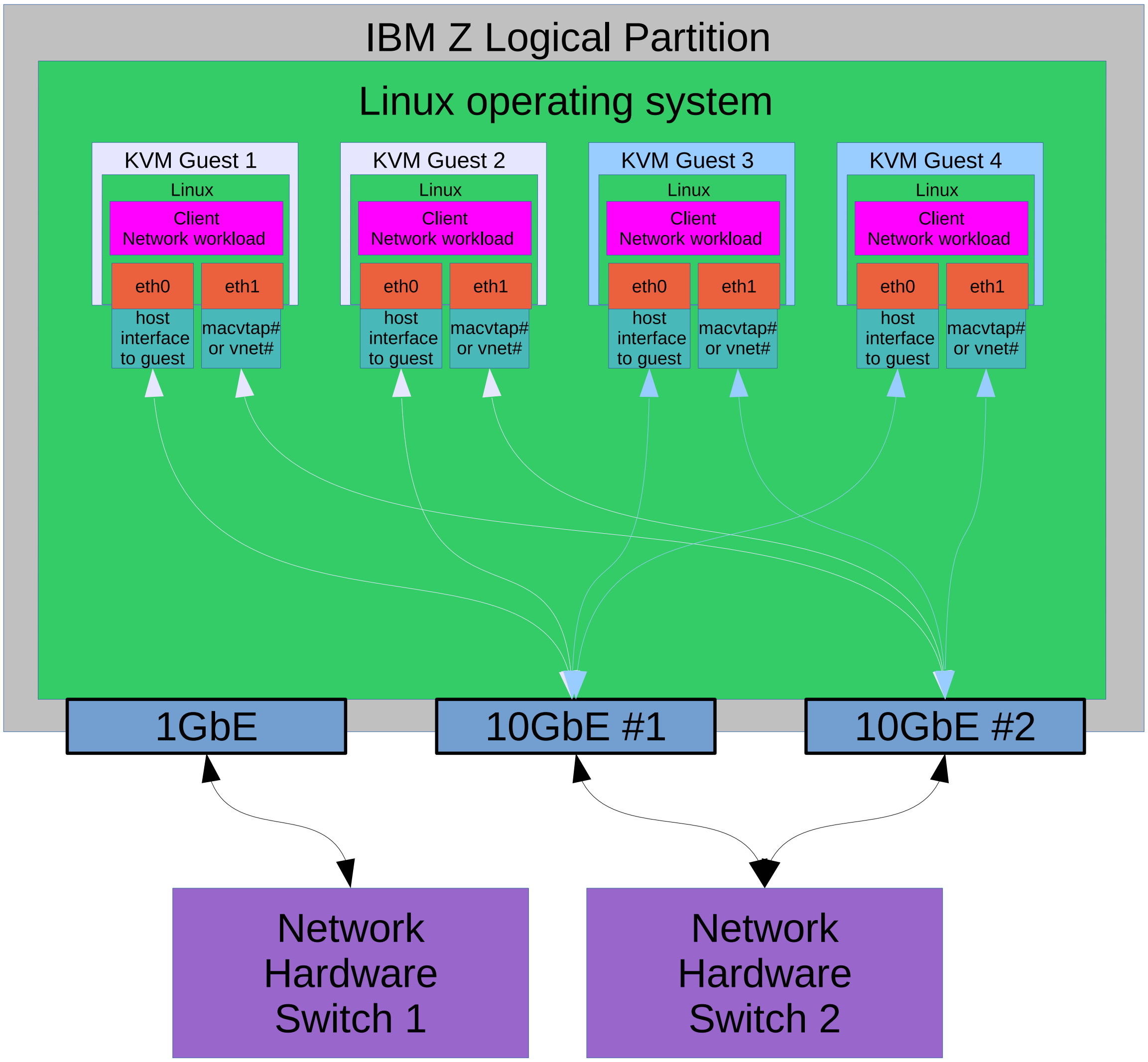


Figure 7. Single KVM host `uperf` pair to KVM guest mappings

KVM guest 1 and 2 (light grey color) are used to form the first `uperf` pair. The next `uperf` pair uses the next sequential KVM guests 3 and 4 (light blue color) and so on.

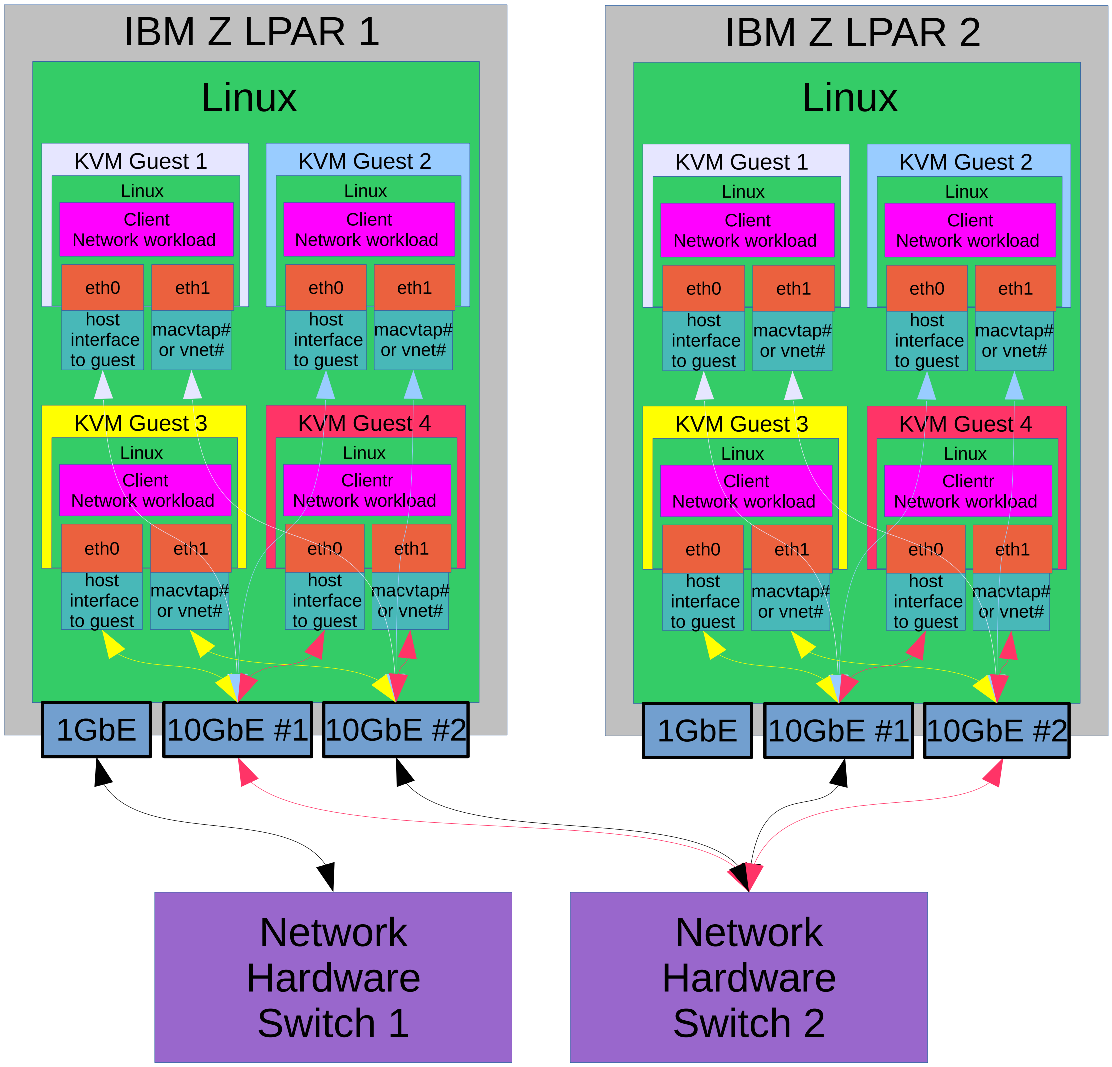


Figure 8. `uperf` pair to KVM guest mappings when multiple KVM hosts are used

For the multiple host configuration, the `uperf` pairs are assembled from a KVM guest residing on a different KVM hosts. Here KVM guest 1 (in light grey) on both hosts are used to form a `uperf` pair. As with the single KVM host configuration, the next KVM guest from each host (in light blue, yellow and red) is used to form the next `uperf` pair and so on. Unlike the single host configuration which handles both clients and servers, the multiple host configurations ends up with all the KVM `uperf` clients running on one KVM host while all the KVM `uperf` servers run on the other. The goal was to spread each `uperf` pair across the KVM hosts.

In the multiple host configuration, for test measurement purposes, each host used a separate network interface which was configured from a separate OSA adapter (see the red connector arrows in Figure 8) to ensure traffic flowed from one LPAR to the other LPAR through the hardware switch, to evaluate the network path and behavior seen when communicating with an external system.

Chapter 5. Testing methodology

This topic describes the testing methodology that was used in this paper.

Testing was performed by running uperf with the 20 unique tests defined in the topic “Workload tests” of Chapter 4, “Network workload,” on page 11, that scaled the simulated users for each of the five workloads. The execution of each of the 20 workload tests invoked sequentially, one after the other, are called a “run”. Runs were performed for 1, 2, 4 and 8 uperf pairs. The KVM host LPARs were freshly rebooted at the start of each run.

A separate run was conducted for each LPAR, network configuration, uperf pair count and when a different value was used for one of the many Linux operating system, network, KVM, or uperf settings / tunables.

Running uperf

To run uperf, the server must first be started with the following command:

```
[root@kvmguest ~] # uperf -s
```

When the server starts, it enters listening mode waiting for one or more clients to connect.

Once the server is listening, the client can then be started with the following command:

```
[root@kvmguest ~] # uperf -m profile.xml -a -i 30
```

The **-m** parameter specifies the XML which contains the workload test definition or profile that uperf will use for this invocation. Two additional command line options were used. First, the **-a** option to tells uperf to collect all statistics which is useful for analysis purposes. Second, the **-i** option specifies the interval (in seconds) until the next real-time update of throughput and operations per second statistics are reported by uperf to the user.

There are a variety of additional command line parameters available. For the complete list of options, refer to the www.uperf.org Web page.

uperf Output

An example of the data generated by the command **# uperf -m profile.xml -i 30** is listed below. Additionally, specifying the **-a** option includes the details from **Group**, **Strand**, **Transaction** (**Txn**), and**Flowop** sections in the results report.

```
# uperf -m profile.xml -a
Starting 5 processes running profile:tcp_rr ...   0.00 seconds
Txn1          0 /   1.00(s) =            0           5op/s
Txn2    18.49MB / 300.50(s) =   516.11Kb/s       64513op/s
Txn3          0 /   0.00(s) =            0           0op/s
------------------------------------------------------------------------

Total   18.49MB / 302.60(s) =   512.52Kb/s       64065op/s

Group Details
------------------------------------------------------------------------
Group0        0 / 302.50(s) =            0           0op/s

Strand Details
------------------------------------------------------------------------
Thr0     3.70MB / 302.50(s) =   102.55Kb/s       12818op/s
Thr0     3.70MB / 302.50(s) =   102.55Kb/s       12818op/s
Thr0     3.70MB / 302.50(s) =   102.55Kb/s       12818op/s
Thr0     3.70MB / 302.50(s) =   102.55Kb/s       12818op/s
Thr0     3.70MB / 302.50(s) =   102.55Kb/s       12818op/s

Txn                Count         avg         cpu         max         min
------------------------------------------------------------------------
Txn0                   5    453.42us      0.00ns    513.58us    383.42us
Txn1             9693177    154.87us      0.00ns    200.06ms     32.28us
Txn2                   5     10.65us      0.00ns     12.38us      8.63us

Flowop             Count         avg         cpu         max         min
------------------------------------------------------------------------
accept                 5    453.10us      0.00ns    513.18us    383.18us
write            9693177      3.53us      0.00ns    232.93us      1.49us
read             9693172    151.25us      0.00ns    200.06ms    867.00ns
disconnect             5     10.50us      0.00ns     12.30us      8.47us

Netstat statistics for this run

------------------------------------------------------------------------

Nic       opkts/s     ipkts/s      obits/s      ibits/s
10gb2           0           0            0     24.75b/s
1gb             1           1     2.42Kb/s    617.18b/s
10gb1       32033       32033    17.17Mb/s    13.58Mb/s
-------------------------------------------------------------------------------------------------

Run Statistics
Hostname            Time       Data   Throughput   Operations   Errors   xferbytes          nsec
-------------------------------------------------------------------------------------------------

10.12.38.254     302.60s    18.49MB   512.52Kb/s   19386369     0.00      19386354   302603373999
master           302.60s    18.49MB   512.52Kb/s   19386364     0.00      19386349   302603147820
-------------------------------------------------------------------------------------------------

Difference(%)     -0.00%     -0.00%        0.00%     -0.00%     0.00%
```

The uperf output data is separated into *three* sections.

The first section provides real-time updates of the execution progress of the transactions defined in the profile. In the example profile above, you will see the three transactions that were defined.

- The first transaction was to connect each process from the client to the server.
- The second transaction sends a 1 byte Request and reads a 1 byte Response. The throughput rate and operations per second for each transaction is updated based on the specified interval (-i) parameter (which defaults to 1 second).

- The third transaction disconnects each client process from the server. When the transaction has completed, a final update is displayed.[5]

If the **-a** parameter is specified, four additional detail sections (group, strand, transaction and flowop) will be displayed. Each section will show totals for each element type relevant to that section.

The next section reports the Netstat statistics for the run, showing the packets and bits sent and received by each network interface in the system. Typically only one or two interfaces will be relevant to the test that was run.

The final section reports the Run statistics which is a high level summary of the completed test, reporting the total test duration, data transferred, operations and errors across the hostname or IP of all the members involved in the test. The data in this section will be a primary source of the results used to conduct our comparisons.

For the measurement results used in this paper, the workload test **throughputs** and **transaction times** were extracted from the `uperf` output report.

[5] Private network: Wikipedia

Chapter 6. KVM Host Networking Configuration Choices

There are many choices for network configuration in the KVM host. In this topic, four choices are discussed.

The four network configuration choices are:
- Using a Linux bridge with NAT for KVM guests
- Using a Linux bridge (without NAT) for KVM guests
- Using an Open vSwitch bridge with KVM guests
- Using the MacVTap driver with KVM guests

Each of these choices has trade-offs. In the topics below, each choice will have a topic describing the configuration, and a "review" that highlights the pros and cons to help you to decide which configuration is best for your needs.

OSA interface traffic forwarding

Before reviewing the various bridge choices, lets first discuss two OSA interface configuration modes that can affect how you choose to incorporate bridges in your network configurations.

OSA MAC address registration

By default, the OSA card only forwards network traffic destined to devices that the OSA device knows about. The OSA "only knows about" devices that are registered with the OSA device. For each registered device, the OSA cards maintains a MAC address entry in the "Forwarding Database" (see **man bridge**) on the KVM host. To list the Forwarding Database entries for the KVM host, use the following command:

```
[root@kvmhost] # bridge fdb show
01:00:5e:00:00:01 dev 10gb2 self permanent
33:33:00:00:00:01 dev 10gb2 self permanent
33:33:ff:c4:11:fd dev 10gb2 self permanent
01:00:5e:00:00:01 dev 10gb1 self permanent
33:33:00:00:00:01 dev 10gb1 self permanent
33:33:ff:c4:11:fe dev 10gb1 self permanent
33:33:00:00:00:01 dev 1gb self permanent
01:00:5e:00:00:01 dev 1gb self permanent
33:33:ff:6b:00:39 dev 1gb self permanent
```

To view the Forwarding Database entries associated to a specific OSA device, use the command:

```
[root@kvmhost] # bridge fdb show dev <interface-name>
```

For example:

```
[root@kvmhost] # bridge fdb show 10gb1
01:00:5e:00:00:01 dev 10gb1 self permanent
33:33:00:00:00:01 dev 10gb1 self permanent
33:33:ff:c4:11:fe dev 10gb1 self permanent
```

Each bridge **fdb** entry contains two relevant pieces of information. The first is the *registered MAC address* of a device in the KVM host, and the second is to *which KVM host interface* that the MAC address is registered.

Before you can register a new device, you must know its MAC address. To list the available devices and their MAC addresses, use the **ifconfig** or **ip link show** command.

To register a new device on the OSA card, use this command:

```
[root@kvmhost] # bridge fdb add <new-device-mac-address> dev <interface-name>
```

Once the MAC address of the target device is known to the OSA interface, the OSA will forward any traffic it receives which is destined for the target device. Additionally, any other devices that are attached to the target device need to be known to the OSA device as well. This includes network interfaces of all KVM guests. Depending on the KVM releases, the libvirt daemon (`libvirtd`) might manage the MAC registration (adds and deletes) for KVM guests using MacVTap devices as they are started and stopped. However, if you decide to configure additional devices between the KVM guests and the OSA interfaces, manual registration on the OSA is required.

MAC address registration on OSA interfaces does not persist across reboots of the KVM host. It will be necessary to perform manual MAC registration each time the KVM host restarts. A better choice might be to create a script that is configured to run at system startup time.

OSA Bridgeport mode

As more complex network configurations are used, the requirement of MAC registration become more complex. For this reason, the firmware of newer OSA cards supports a new configuration option called Bridgeport. Bridgeport is an OSA specific feature that activates promiscuous mode on the OSA adapters. Bridgeport mode, when enabled, disables packet address inspection and filtering and causes the OSA interface to forward traffic with unknown destinations to all attached devices (e.g. traffic destined to other software bridges, switches or interfaces running in the KVM host).

Bridgeport essentially disables the requirement for OSA MAC address registration that was previously described.

To view the configuration of an OSA interface, use this command:

```
[root@kvmhost] # lsqeth <interface-name>
```

For example:

```
[root@kvmhost] # lsqeth 10gb1
Device name : private1
-------------------------------------------------------------------------
card_type : OSD_10GIG
cdev0 : 0.0.e000
cdev1 : 0.0.e001
cdev2 : 0.0.e002
chpid : 84
online : 1
portname : no portname required
portno : 0
state : UP (LAN ONLINE)
priority_queueing : always queue 2
buffer_count : 128
layer2 : 1
isolation : none
bridge_role : none
bridge_state : inactive
bridge_hostnotify : 0
bridge_reflect_promisc : none
switch_attrs : unknown
```

The value of the field `bridge_reflect_promisc` reports the state of Bridgeport mode.

Note: If the field `bridge_reflect_promisc` is not present, then Bridgeport mode may not be supported by either the:

- OSA adapter in the system.
- version of KVM being used.

To enable Bridgeport mode, do the following:

- Enable the OSA `bridge_reflect_promisc` on the OSA:

```
[root@kvmhost] # echo "primary" > /sys/class/net/<interface-name>/device/bridge_reflect_promisc
```

- enable promiscuous mode in the Linux Kernel-based:

```
[root@kvmhost] # ip link set dev <interface-name> promisc on
```

With Bridgeport active, device MAC address registration is no longer required.

Note: If an OSA adapter is shared across multiple LPARs on the same system, only a single LPAR can be configured for Bridgeport mode at any point in time. Separate LPARs being configured for promiscuous mode concurrently require separate OSA adapters.

KVM default NAT-based networking

NAT-based networking is commonly provided and enabled as default by most major Linux distributions that support KVM virtualization.

This network configuration uses a Linux bridge in combination with Network Address Translation (NAT) to enable a guest OS to get outbound connectivity regardless of the type of networking (wired, wireless, dial-up, and so on) used in the KVM host without requiring any specific administrator configuration.

Like other software bridge choices, NAT-based networking allows KVM guests sharing the same bridge to communicate together even if the bridge is not connected to an interface in the KVM host or if the KVM host has no physical networking installed or enabled.

While NAT is a convenient choice that is extremely flexible, allowing a guest OS to easily connect to the world outside of the KVM host, it has characteristics which can make it more or less desirable for many business and enterprise uses.

First, by default, the bridge used with NAT-based connectivity is typically configured to use private IP addresses from a `192.168.x.x` subnet. "Addresses in the private space are not allocated to any specific organization and anyone may use these addresses without approval from a regional Internet registry."[6] Using a `192.168.x.x` subnet allows a Linux distribution to avoid many of the configuration tasks and complexities regarding network resource reservation and administration, which is extremely convenient.

A second characteristic or restriction with using the KVM default NAT-based networking is that interfaces associated to the NAT are not, by default, visible outside of the KVM host running the NAT. This mean that external systems and their networking components have no knowledge of or way to route network traffic directly to a KVM guest OS on separate KVM host. This generally means a NAT would not be unusable for server workloads that rely on receiving unsolicited external network requests in order to do work.

A third characteristic, the use of Network Address Translation in addition to a software bridge, creates additional overhead which can affect network performance throughput and latency as well as potentially increases the consumption of CPU and memory resources. NAT behavior is normally implemented using a linux firewall that employs static and dynamic firewall rules. The use of the firewall puts additional demands on system.

For most Linux distributions, NAT-based networking is configured and available by default when the operating system is installed. Typically, the name for the default NAT bridge is `virbr0` and the typical name for the default network is `default`.

To list which networks have been defined to the libvirt daemon for use by KVM guests, use the following command:

```
[root@kvmhost ~] # virsh net-list
Name    State  Autostart Persistent
----------------------------------------------------------
default active yes       yes
```

To use the default NAT bridge by a KVM guest, add or edit the network section of the libvirt XML configuration file for the KVM guest to include the name of the default bridge:

```
<interface type="bridge">
    <source bridge="bridge-name"/>
    <model type="virtio"/>
    <driver name="vhost"/>
</interface>
```

[6] Private network: Wikipedia

To define a new NAT bridge, complete the following steps.

1. Create a new libvirt network configuration like the following:

   ```
   [root@kvmhost ~] # vi ~/new-kvm-network.xml
   <network>
       <name>newnatnetwork</name>
       <forward mode='nat'>
           <nat>
               <port start='1024' end='65535'/>
           </nat>
       </forward>
       <bridge name='my-bridge-name' stp='on' delay='0'/>
       <ip address='192.168.X.1' netmask='255.255.255.0'>
           <dhcp>
               <range start='192.168.X.2' end='192.168.X.254'/>
           </dhcp>
       </ip>
   </network>
   ```

   Change the `<name>`, `<bridge name>` and `<ip address>` to suite your needs. It is recommended that you choose names and ip address that are different from the default bridge to avoid conflicts. The typical ip address of the default network is 192.168.122.1.

   The network name defined in network XML `<name>` tag is reported in the `Name` field using the **`virsh net-list`** command.
2. Add the new network definition XML file to libvirt:

   ```
   [root@kvmhost ~] # virsh net-create ~/net-nat-network.xml
   ```

   Once added to libvirt, the new network definition will *persist*.
3. To set the new network to automatically startup each time the KVM host is rebooted, do this:

   ```
   [root@kvmhost ~] # virsh net-autostart <network-name-from-xml>
   ```

   Specify the network by the `<name>` defined in the XML file.
4. Add or change the KVM guest's configuration to use this network or bridge name.

   To view configuration details of a specific network defined in libvirt, use the following command:

   ```
   [root@kvmhost ~] # virsh net-dumpxml<libvirt-network-name>
   ```

Using a Linux Bridge

An alternative to using a NAT-based network would be to use a standard Linux network bridge.

A network bridge is a Link Layer device which forwards traffic between networks based on MAC addresses and is therefore also referred to as a Layer 2 device. It makes forwarding decisions based on tables of MAC addresses which it builds by learning what hosts are connected to each network. A software bridge can be used within a Linux host in order to emulate a hardware bridge, for example in virtualization applications for sharing a NIC with one or more virtual NICs.[7]

[7] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s2-networkscripts-interfaces_network-bridge.html

In the context of KVM, a Linux bridge is used to connect the KVM guest interface to a KVM host network interface.

To create a Linux Bridge on a KVM host, use the following command:

```
[root@kvmhost ~] # brctl addbr <bridge-name>
```

Next, if KVM guest interfaces that will be connecting to the bridge need to communicate with other systems external to the KVM host, the Linux bridge must be instructed which KVM host interface to use. To attach a Linux bridge to an OSA interface in the KVM host, use the following command:

```
[root@kvmhost ~] # brctl addif <bridge-name> <host-interface-name>
```

If the bridge needs to send or receive network traffic from external systems, changes in the KVM host will need to be configured to enable this. As discussed in “OSA interface traffic forwarding” on page 21, there are two ways to do this:

- By registering the bridge device with the OSA card.
- By configuring the OSA card for Bridgeport mode and enabling promiscuous mode in the KVM host kernel.

To configure a KVM guest interface to use a Linux bridge, the XML stanza to define the interface for the KVM guest in the libvirt XML configuration file should include the following:

```
<interface type="bridge">
    <mac address="11:22:33:44:55:66"/>
    <source bridge="bridge-name"/>
    <model type="virtio"/>
    <driver name="vhost"/>
</interface>
```

For a KVM guest interface to connect to a bridge, use `<interface type="bridge">` and specify the name of the bridge using the `<source bridge=` keyword.

For better performance throughput and latency, it is recommended that KVM guests use the newer `vhost-net` driver, rather than the older para-virtualized `virtio-net` driver, by specifying the keyword `<driver name="vhost"/>` in the guest's libvirt configuration file. `vhost-net` uses an in-kernel guest networking performance enhancement which moves network packets between the guest and the host system using the Linux kernel rather than QEMU. This avoids context switches from the kernel to user space to improve overall performance.

The MAC address field is optional and if omitted, the libvirt daemon will generate a unique value.

After the changes have been saved in the libvirt XML configuration file for the KVM guest, the libvirt daemon needs to be informed by using the following commands:

```
[root@kvmhost ~] # virsh undefine <kvm-guest-name>
[root@kvmhost ~] # virsh define <kvm-guest-libvirt-xml-file>
```

And finally, remember to restart the KVM guest for the changes to take affect.

Using Open vSwitch

Open vSwitch (abbreviated to *OVS*) is a production quality, multilayer virtual switch. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (for example, NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag).

In addition, it is designed to support distribution across multiple physical servers similar to VMware's vNetwork distributed vswitch or Cisco's Nexus 1000V. Also see the *full feature list*.[8]

Open vSwitch has many features that can not be found in the standard Linux software bridges. If your configuration needs the functionality provided by these features, Open vSwitch may be your best choice. In this comparison Open vSwitch was tested as an alternative to the Linux software bridge, therefore none of the additional capabilities were configured or tested.

To use Open vSwitch, the service must be enabled and started using the following commands:

```
[root@kvmhost ~] # systemctl enable openvswitch.service
[root@kvmhost ~] # systemctl start openvswitch.service
```

Enablement and starting need only be done once and will persist across KVM host restarts.

To create an Open vSwitch bridge use:

```
[root@kvmhost ~] # ovs-vsctl add-br <bridge-name>
```

Next, if KVM guest interfaces that will be connecting to the bridge need to communicate with other systems external to the KVM host, the Open vSwitch bridge must know which KVM host interface to use. Attach the OSA device associated with the KVM host interface to the Open vSwitch bridge by connecting them together using an Open vSwitch bridge port.

```
[root@kvmhost ~] # ovs-vsctl add-port <bridge-name> <host-interface-name>
```

Once the bridge exists and if the bridge needs to send or receive network traffic outside of the KVM host, the KVM host will need to be configured to forward traffic to bridge. As discussed in “OSA interface traffic forwarding” on page 21, there are two possible ways to do this:

- By registering the bridge device with the OSA card.
- By configuring the OSA card for Bridgeport mode and enabling promiscuous mode in the KVM host.

To configure a KVM guest interface to use an Open vSwitch bridge, the XML stanza to define the KVM guest's interface in the guest's libvirt XML configuration file should include the following:

[8] http://openvswitch.org/

```
<interface type="bridge">
    <mac address="11:22:33:44:55:66"/>
    <source bridge="bridge-name"/>
    <virtualport type="openvswitch">
        <parameters interfaceid="<ovs-bridge-uuid>"/>
    </virtualport>
    <model type="virtio"/>
    <driver name="vhost"/>
</interface>
```

For the KVM guest interface to connect to a bridge, use **<interface type="bridge">** and specify the name of the bridge using the keyword **<source bridge=**.

For better performance throughput and latency, it is recommended that KVM guests use the newer vhost-net driver, rather than the older paravirtualizated **virtio-net** driver, by specifying the keyword **<driver name="vhost"/>** in the guest's libvirt configuration file. Vhost-net uses an in-kernel guest networking performance enhancement which moves network packets between the guest and the host system using the Linux kernel rather than QEMU. This avoids context switches from the kernel to user space to improve overall performance.

When using an Open vSwitch bridge, it is required to not only specify the bridge by name (**<source bridge=**) but to also specify the identity of OVS bridge using it's UUID in the **<parameters interface=** tag within the **<virtualport type=** stanza. The UUID of the OVS bridge can be obtained using the following command:

```
[root@kvmhost ~] # ovs-vsctl show
2dbde39b-9f37-4a73-a82e-8afeaf723fb6
ovs_version: "2.0.1"
```

The MAC address field is optional and if omitted, the libvirt daemon will generate a unique address.

After the changes have been saved in the KVM guest's libvirt XML configuration file, the libvirt daemon needs to be informed, using the following commands:

```
[root@kvmhost ~] # virsh undefine <kvm-guest-name>
[root@kvmhost ~] # virsh define <kvm-guest-libvirt-xml-file>
```

And lastly, remember to restart the KVM guest for the changes to take affect.

Using the MacVTap driver

Another alternative to using a bridge to enable a KVM guest to communicate externally is to use the Linux MacVTap driver.

"Macvtap is a new device driver meant to simplify virtualized bridged networking. It replaces the combination of the tun/tap and bridge drivers with a single module based on the macvlan device driver. A macvtap endpoint is a character device that largely follows the tun/tap ioctl interface and can be used directly by kvm/qemu and other hypervisors that support the tun/tap interface. The endpoint extends an existing network interface, the lower device, and has its own mac address on the same ethernet segment. Typically, this is used to make both the guest and the host show up directly on the switch to which the host is connected.[9]

A key difference between using a bridge is that MacVTap connects directly to the network interface in the KVM host. This direct connection effectively shortens the codepath, by bypassing much of the code and components in the KVM host associated with connecting to and using a software bridge. This shorter codepath usually improves throughput and reduces latencies to external systems.

MacVTap modes

MacVTap can be configured in any of three different modes which determine how the macvtap device communicates with the lower device in the KVM host. The three possible modes are VEPA, Bridge and private mode.

1. Virtual Ethernet Port Aggregator (**VEPA**) is the default mode. Data flows from one endpoint down through the source device in the KVM host out to the external switch. If the switch supports hairpin mode, the data is sent back to the source device in the KVM host and from there sent to the destination endpoint.

   Most switches today do not support hairpin mode, so the two endpoints are not able to exchange ethernet frames, although they might still be able to communicate using an tcp/ip router. A linux host used as the adjacent bridge can be put into *hairpin* mode by writing to */sys/class/net/***dev***/brif/***port***/ hairpin_mode*. This mode is particularly interesting for data-centers and cloud provides where the ability to manage virtual machine networking at the switch level is desirable. Switches aware of the VEPA guests can enforce filtering and bandwidth limits per MAC address without the Linux host knowing about it.
2. **Bridge**, connecting all endpoints directly to each other. Two endpoints that are both in bridge mode can exchange frames directly, without the round trip through the external bridge. This is the most useful mode for setups with classic switches, and when inter-guest communication is performance critical. (Figure 9 on page 30 shows the communication path using **Bridge** mode.)
3. For completeness, a **private** mode exists that behaves like a VEPA mode endpoint in the absence of a hairpin aware switch. Even when the switch is in hairpin mode, a private endpoint can never communicate to any other endpoint on the same lowerdev.[10]

[9] MacVTap - Linux Virtualization Wiki

[10] MacVTap - Linux Virtualization Wiki

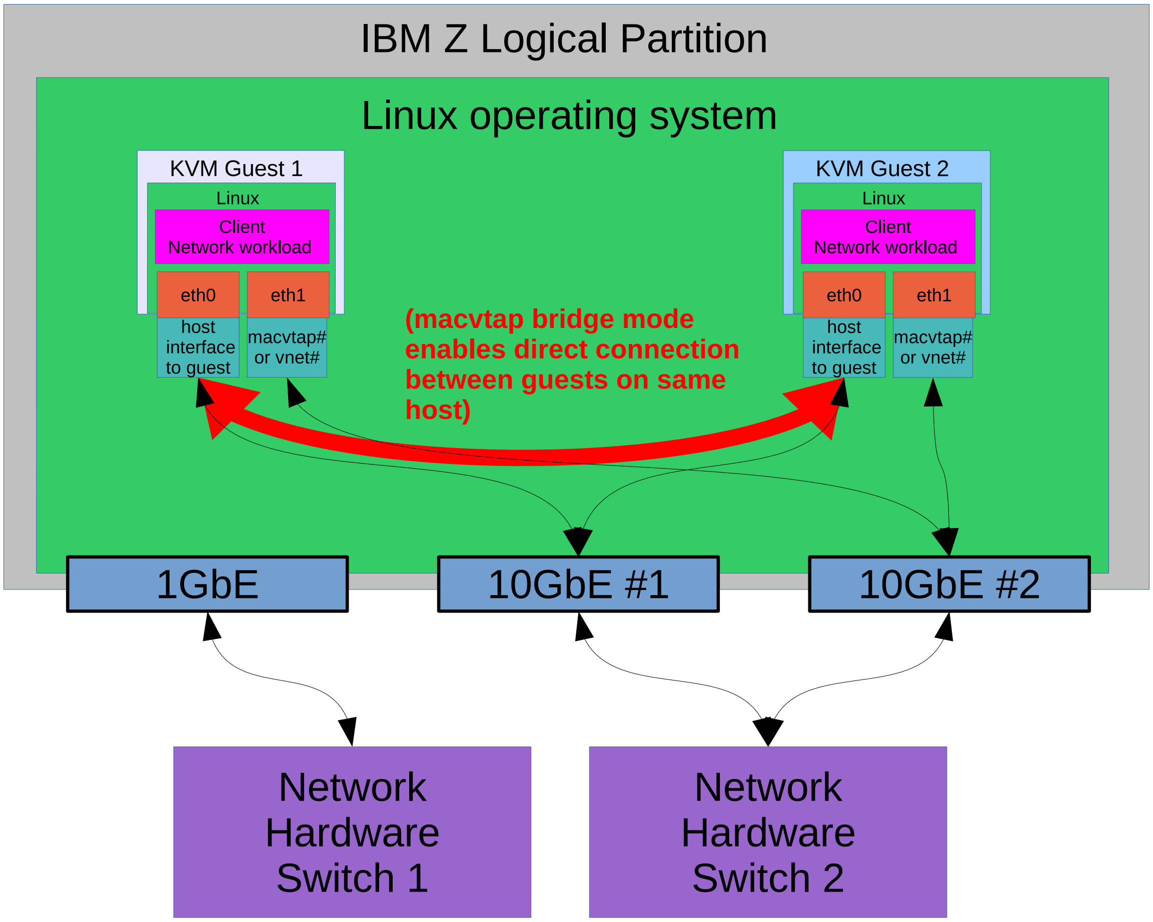


Figure 9. Single LPAR using MacVTap bridge mode for direct connection between KVM guests

MacVTap isolation / limitations

Each of these modes provide different levels of endpoint isolation. Consider your needs when choosing which macvtap mode is right for your environment.

Bridge mode provides the greatest degree of flexibility enabling inter-guest communication within a single KVM host.

VEPA mode, without a network switch thats supports *hairpin* mode, prevents any endpoints using the same KVM host interface from being able to communicate.

In **Private** mode, even with a network switch thats supports *hairpin* mode, a KVM guest endpoint will not be able to communicate with its KVM host using the same lower level source device.

Without a switch that supports *hairpin* mode, KVM guests configured to use Bridge or VEPA modes will not be able to directly communicate with the KVM host using the same KVM host interface. This limitation can be overcome if the KVM host has multiple interfaces using different ethernet segments (subnets).

To configure a KVM guest interface to use the MacVTap driver, the XML stanza to define the interface for KVM guest the in the guest's libvirt XML configuration file should include the following:

```
<interface type="direct">
    <mac address="12:34:56:78:9a:bc"/>
    <source dev="private1" mode="bridge"/>
    <model type="virtio"/>
    <driver name="vhost"/>
</interface>
```

The relevant XML tags here are:

- `<interface type="direct">`
- `<source dev="kvm-host-device" mode="bridge">`

The **interface type** for macvtap is **direct**, which specifies a direct mapping to an existing KVM host device. The **source dev** keyword specifies the KVM host network interface name that will be used by the macvtap interface for the KVM guest. The **mode** keyword defines which macvtap mode will be used.

The MAC address field is optional and if omitted, the libvirt daemon will generate a unique address.

After the changes have been saved in the KVM guest's libvirt XML configuration file, the libvirt daemon needs to be informed, using the following commands:

```
[root@kvmhost ~] # virsh undefine <kvm-guest-name>
[root@kvmhost ~] # virsh define <kvm-guest-libvirt-xml-file>
```

And lastly, remember to restart the KVM guest for the changes to take affect. The defined macvtap interface will be persistent and automatically started whenever the KVM guest restarts.

All of MacVTap testing done in this paper used the macvtap **mode="bridge"** which allowed KVM guests on the same KVM host attached to the same KVM host network interface to communicate with each other. This enabled KVM guests on the same KVM host with `uperf` client and server roles to be paired.

Network Configuration Pros and Cons

In this topic we discuss the advantages and disadvantages of the three evaluated Network Configurations.

Which of the tested Networking Configurations is the appropriate choice for you? That depends on a number of factors, like highest throughputs, less CPU consumption, better CPU efficiency, easier configuration, usability, isolation and more.

To help compare and contrast characteristics and behavior of the tested network configurations, graphs have been produced comparing results from two separate workload runs. Recall that each run consists of 20 different tests (described in "Workload tests" in Chapter 4, "Network workload," on page 11). Each graph compares the 20 workload test results from one run to the results of a separate run.

The graphs show the differences between the two compared runs. In the graphs below the **20 different workload tests** are represented on the X-axis and the **difference** (in %) of 4 different performance characteristics are shown on the

Y-axis. All of the performance characteristics have been normalized to show the difference of a second run (B) when compared to a first run (A). The titles of each graph take the form of “run B config” vs “run A config”.

- If the result from run B is better than run A, the bar on the graph will be positive.
- If run B's result is worse than run A's result, the bar will be negative. In all the graphs a positive bar means a better result.

For throughput, (the **blue** bar), a better results means a higher absolute value.

For transaction times (the **orange** bars), better equates to shorter time. Transaction time is equivalent to the overall latency of network transaction. The faster (or shorter) amount of time a transaction completes the better, so better is less overall time.

The **yellow** and **green** bars represent CPU efficiency for the KVM guest running the `uperf` client (**yellow**) and KVM guest running the `uperf` server (**green**). These bars compare the amount of CPU required to transfer a megabyte of data. For CPU efficiency, a better result or a positive bar means less CPU was required to transfer the same amount of data.

Note: All of the graphs that follow are from test results collected from runs using 4 `uperf` pairs (see “`uperf` pairs” in Chapter 4, “Network workload,” on page 11) which use 8 KVM guests running on 1 or 2 LPAR configurations.

MacVTap driver considerations

From purely a performance perspective, based on the workloads tested and the Linux and KVM levels measured, the MacVTap driver consistently demonstrated higher throughputs and better CPU efficiency.

The MacVTap driver provides exceptional transactional throughput and operations/sec results (up to 10-50%) better than either of the two software bridges. Additionally, throughput of MacVTap scales up with load more quickly compared to using a software bridge. This means that MacVTap is more CPU efficient, consuming less CPU resources to complete the same amount of work. Stated another way, MacVTap can do more work using the same amount of CPU resources.

Although MacVTap is the best performing, it suffers from a couple of issues that may limit the use cases where it would be a suitable choice.

The first limitation is that MacVTap can not readily enable network communication between the KVM host and any of the KVM guests using MacVTap.

- This issue can be overcome in two different ways. The first way to avoid this limitation is to use a special hardware switch that supports *hairpin* mode to connect the IBM Z system to the outside world. However, hairpin mode is not a common feature in most hardware switches and those switches that do have this feature tend to be significantly more expensive.
- The second way to enable KVM host to guest communications is by having multiple network interfaces in the KVM host. Configure the second KVM host interface on the same segment with a different subnet from the first host interface. MacVTap only restricts traffic flow to the same subnet shared between host and guest. While this method works w/o purchasing additional costly

hardware, it still requires that a second interface be available and appropriately configured in the KVM host and each KVM guest.

A second limitation of MacVTap is that it must attach to a physical host interface. MacVTap, unlike software bridges, provides no way to enable KVM guests to communicate without first being attached to a host interface which is active and externally facing. In other words, KVM guests using MacVTap will be external facing and exposed to external network traffic. This is not necessarily a bad thing. It just doesn't provide KVM host only isolation and connectivity for KVM guests that other choices allow.

Open vSwitch considerations

Based on the measurement data collected in the scope of this paper, Open vSwitch is a good choice when the restrictions of MacVTap are undesirable.

From a performance perspective, Open vSwitch tends to trail behind MacVTap in latency sensitive transactional tests and lightly loaded streaming tests. Open vSwitch, however, typically performs as good or better than a standard Linux bridge. In configurations that desire or require a software bridge, Open vSwitch is a good choice.

Open vSwitch is a very sophisticated and complex network component supporting many more features than does a linux bridge. The comparisons done in this paper utilize relatively basic configuration choices to provide network connectivity for KVM guests to communicate with other guests on the same KVM host or to other external systems. If your configuration requires any of the advanced features provided by Open vSwitch, leveraging these capabilities could easily be deemed a higher priority than the uplift in performance provided by MacVTap.

The following topics describe the high level comparison of relative performance between Open vSwitch and MacVTap.

KVM guests and `uperf` pairs running on the same KVM host using a small MTU size

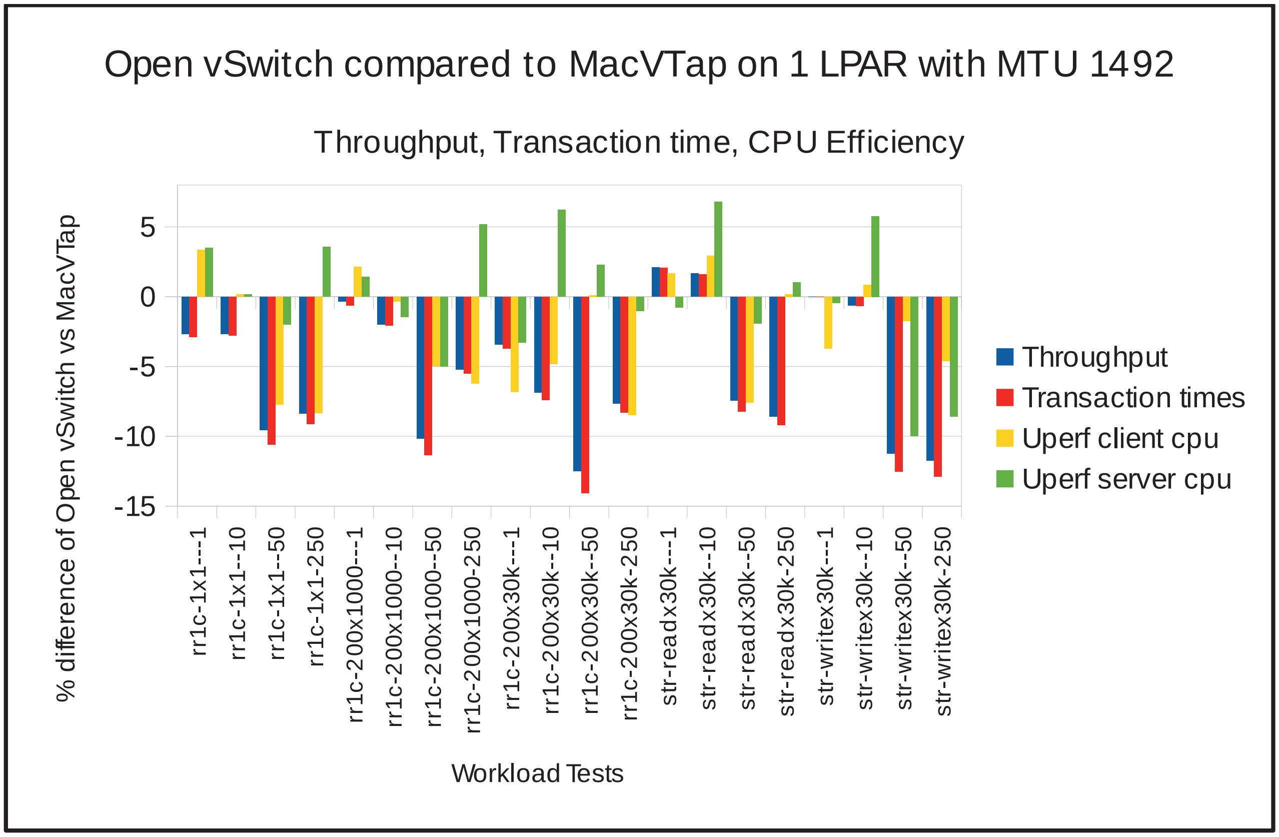


Figure 10. Open vSwitch vs MacVTap with MTU size 1492 on a single KVM host LPAR

Transactional and Streaming performance observations:

- Open vSwitch has higher latencies which lowers the operations/sec which directly impacts throughput by as much as 12% in tests using small, medium or large payload sizes.
- Tests using higher user load (connection counts) see a greater impact.
- CPU consumption for `uperf` client and server tends to vary a fair amount, in this case by +/-7%. The trend for the `uperf` client averages at almost 0%, while the `uperf` server tends to show more CPU consumption for Open vSwitch compared to MacVTap.

Conclusion:

- MacVTap demonstrates higher throughput and lower latency than Open vSwitch.
- MacVTap, especially for the `uperf` server, consumes less CPU generally than does Open vSwitch.
- For these reasons, MacVTap is recommended ahead of Open vSwitch for this LPAR configuration and MTU size combination.

KVM guests and `uperf` pairs running on the same KVM host using a large MTU size

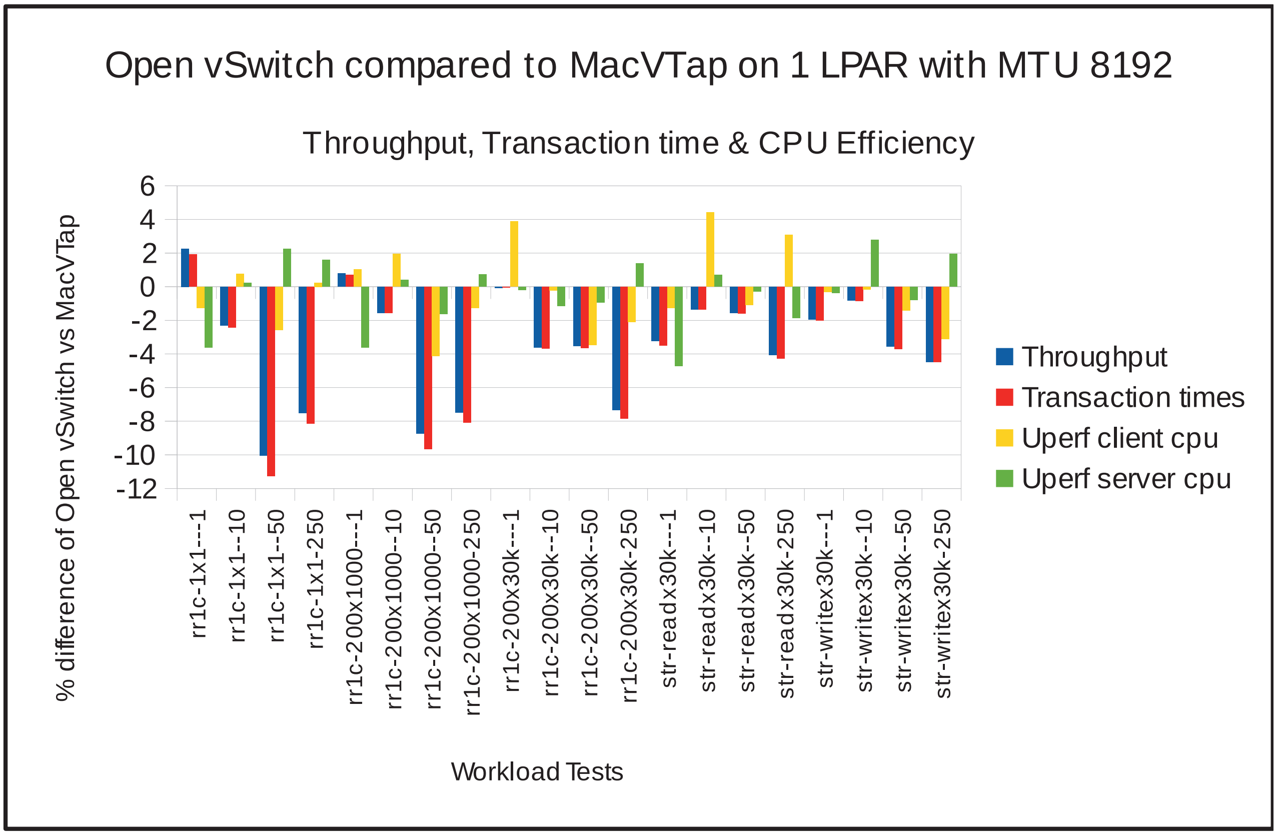


Figure 11. Open vSwitch vs MacVTap with large MTU size on a single KVM host LPAR

Transactional performance observations:

- Open vSwitch has higher throughput and latencies impacts by as much as 12% in tests using small, medium or large payload sizes.
- Greater impacts observed at high user (connection) counts.

Streaming performance observations:

- Has similar trends as the transactional workload tests.
- However the benefits of a larger MTU size (showing a reduced % difference vs MacVTap) can be observed for the tests that have larger payload sizes compared to (transactional) workloads having smaller payload sizes.
- Also, trend showing the performance impacts at higher user loads (connections counts) continues to be seen.

Conclusion:

- MacVTap has better throughput and transaction times compared to Open vSwitch.
- CPU consumption is generally equivalent between MacVTap and Open vSwitch and within the run to run variation typically seen across `uperf` runs.
- Therefore, MacVTap is recommended over Open vSwitch for this LPAR configuration using larger MTU sizes.

KVM guests and `uperf` pairs running across separate KVM hosts using a small MTU size

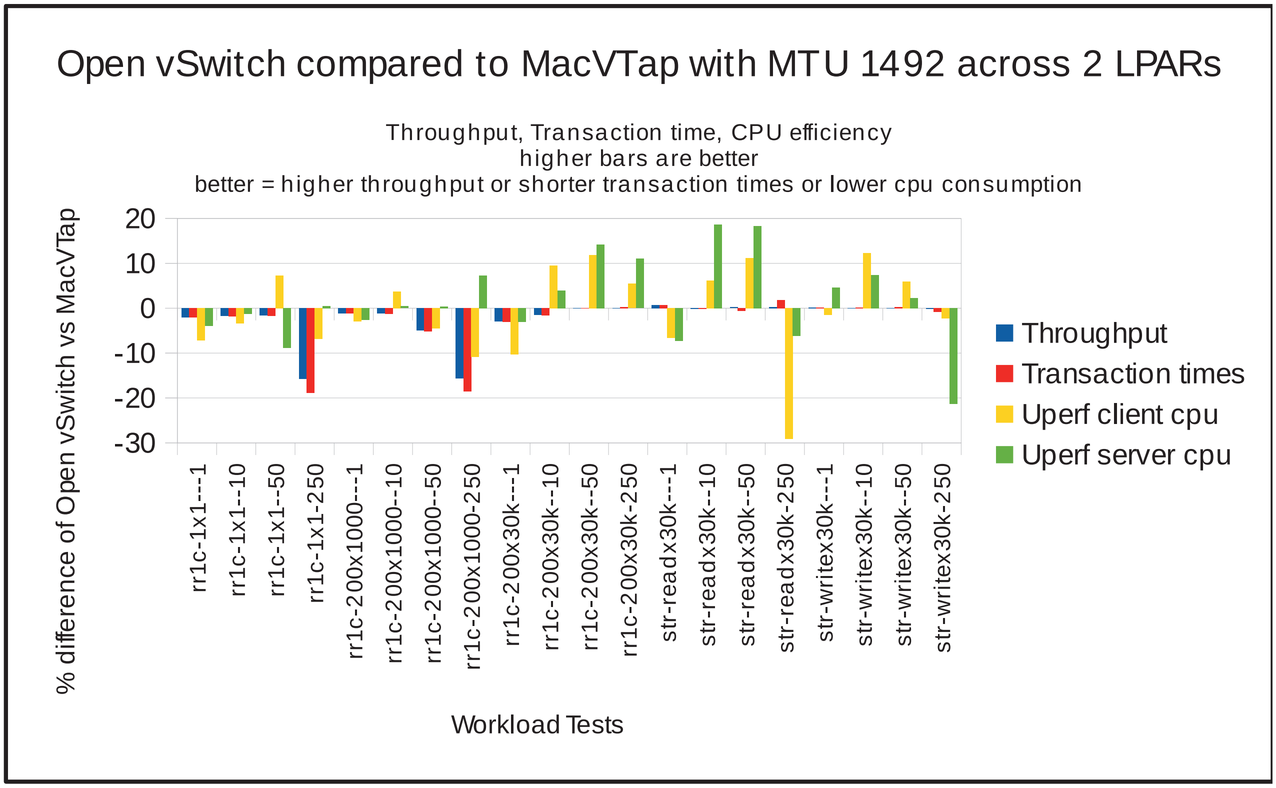


Figure 12. Open vSwitch compared to MacVTap with the small MTU size across LPARs

Transactional performance observations:

- For most workload tests the throughput and latency of Open vSwitch is similar to MacVTap.
- At the highest load levels (250 users), the latencies differences with MacVTap results in Open vSwitch being up to 15% slower.

Streaming performance observations:

- Throughput is essentially the same since it is limited by line speed of the interfaces used in each KVM host.
- Results indicate that Open vSwitch may offer some CPU consumptions savings compared to MacVTap for the `uperf` server, especially for tests with larger payload sizes.

Conclusion:

- For highly concurrent transactional traffic, MacVTap demonstrates up to 15% better throughput and latency than Open vSwitch.
- For streaming workloads (which are throughput limited by the speed of the network interfaces), Open vSwitch has CPU savings advantages over MacVTap.
- MacVTap is recommended for **highlytransactional** workloads
- Open vSwitch is recommended for workloads that have **primarily streaming** behavior.

KVM guests and `uperf` pairs running across separate KVM hosts using a large MTU size

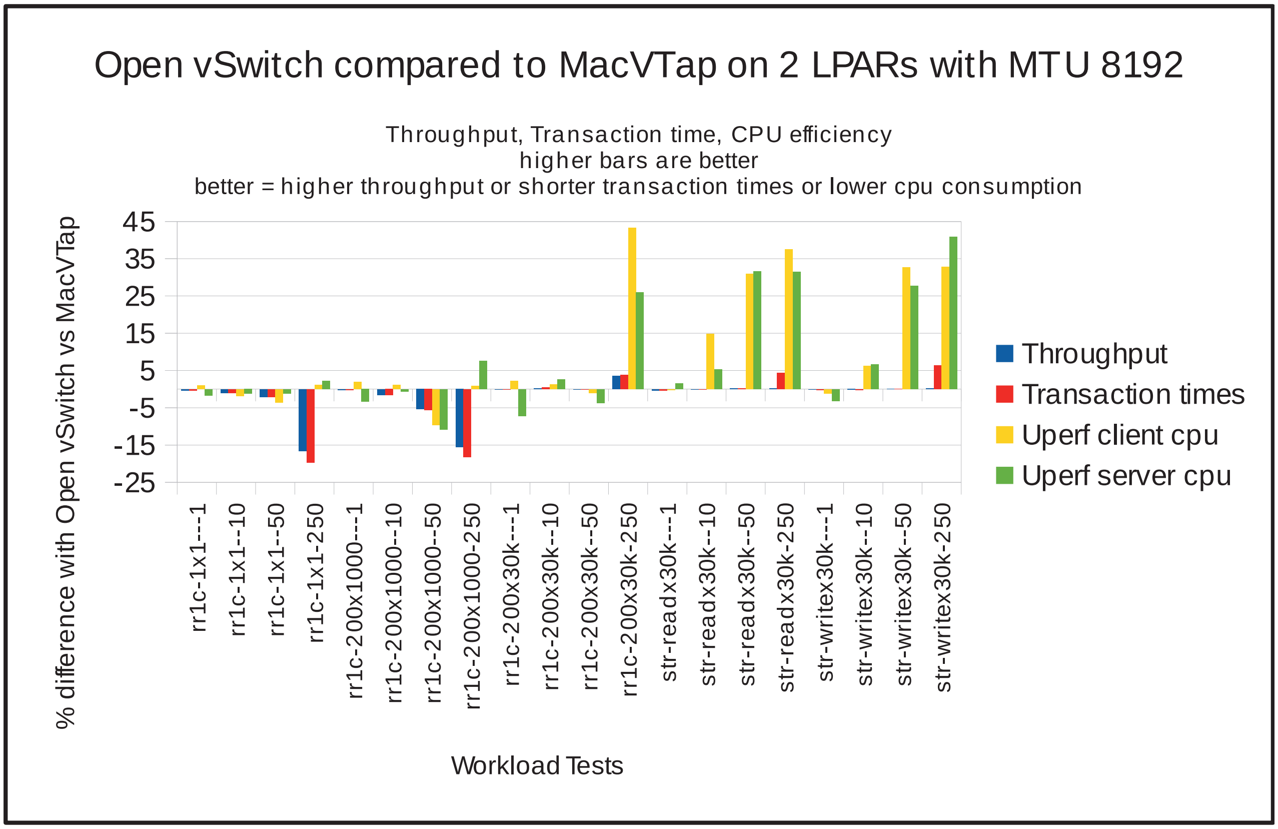


Figure 13. Open vSwitch compared to MacVTap with larger MTU size across LPARs

Transactional performance observation:

- Similar trends (throughput dropping at 250 user load per `uperf` pair) were observed as with the smaller MTU size.

Streaming performance observation:

- Open vSwitch shows a trend of using up to 30+% less CPU than MacVTap.

Conclusion:

- Behavior is very similar to the 2 LPAR configuration with a small MTU size.
- For highly concurrent transactional traffic, MacVTap demonstrates up to 15% better throughput and latency than Open vSwitch.
- For streaming workloads (which are throughput limited by the speed of the network interfaces), Open vSwitch demonstrates CPU savings advantages up to 30% better than MacVTap.
- MacVTap is recommended for **highlytransactional** workloads
- Open vSwitch is recommended for workloads that have **primarily streaming** behavior.

Overall conclusions of Open vSwitch vs MacVTap in our tests

For a single LPAR:
MacVTap is recommended over Open vSwitch because MacVTap achieved higher throughput, lower transaction times and generally less CPU consumption.

For multiple LPARs (which models behavior to external systems):

- MacVTap is recommended for highly transactional workloads.
- Open vSwitch is recommended for primarily streaming behavior.

For either 1 or 2 LPARs:
Using a larger MTU size achieves higher throughputs and lower latencies than smaller MTU size and is recommended choice.

Other considerations

- Open vSwitch does not require any special hardware support (ie. no switch with *hairpin* mode required) to enable the KVM host and KVM guests to communicate directly.
- Open vSwitch can be configured to provide a KVM host isolation mode. Unlike MacVTap, Open vSwitch does not require being attached to a KVM host interface in order to operate, providing a pure virtual and isolated network.
- By only connecting KVM guests to an Open vSwitch bridge and not the host interface, the KVM guests can communicate with each other and the KVM host while being detached and isolated from all network traffic originating from or destined to go outside of the KVM host environment.

Linux bridge

Within the context of the workload analysis and measurement results obtained, the standard software bridge included in Linux can also be a reasonable choice for KVM guest connectivity.

Linux bridges avoids the same restrictions imposed by the MacVTap driver. The Linux bridge provides performance characteristics that are equivalent to or within 10% of Open vSwitch performance results.

The following topics describe the high level comparison of relative performance between the Linux bridge and Open vSwitch.

KVM guests and `uperf` pairs running on the same KVM host using a small MTU size

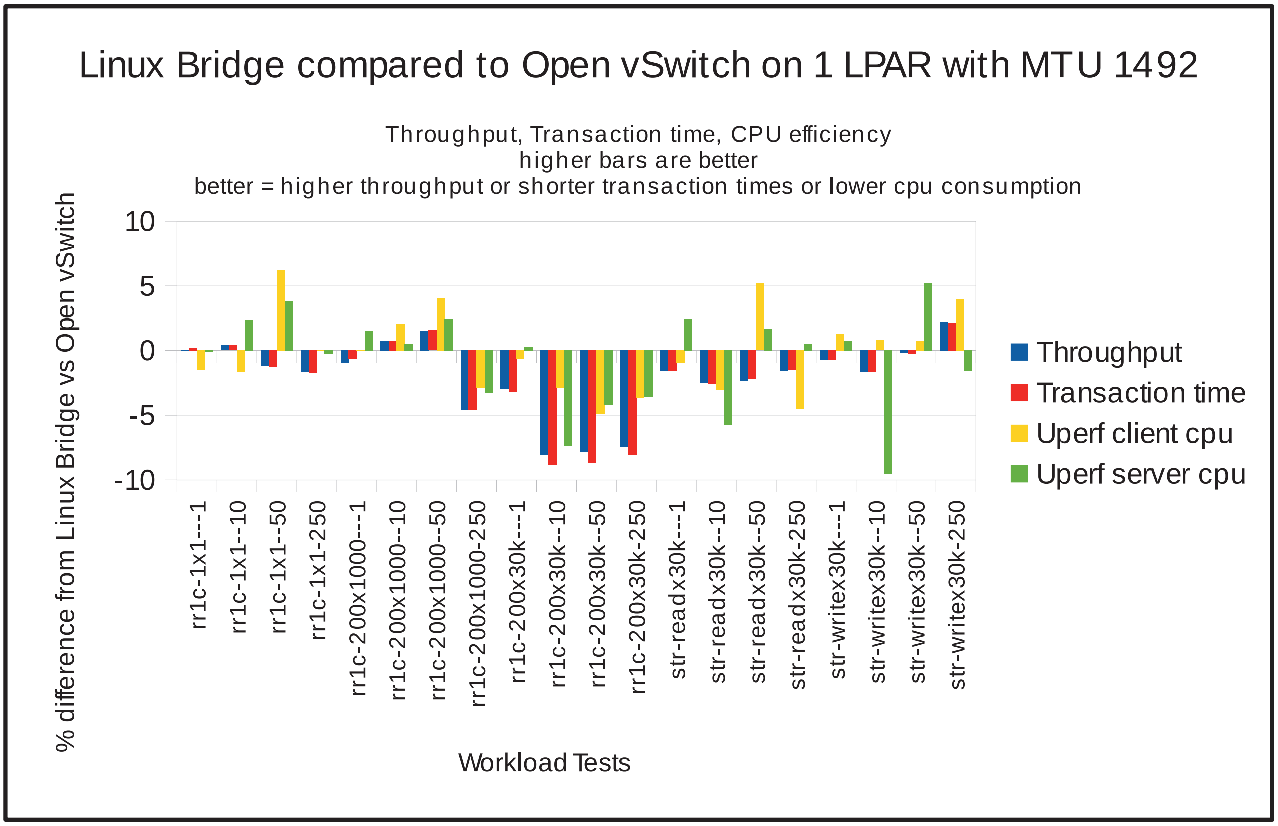


Figure 14. Single LPAR Linux Bridge compared to Open vSwitch with small MTU size

Transaction performance observations:

- The latency and throughput results at the small to medium payload sizes are within measurement variations compared to Open vSwitch.
- At small and medium payload sizes, the Linux bridge may have very small CPU consumption savings.
- As the bandwidth load becomes high, the Linux bridge performance tends to drop off up to ~7% versus Open vSwitch.

Streaming performance observations:

- Throughput and latency performance are nearly equivalent to Open vSwitch, hovering within 2% (which is within the normal measurement variations) with a trend to a very slight drop.

Conclusion:

- For small and large payload sizes, the Linux bridge is essentially equivalent to Open vSwitch.
- For moderate size payloads, Open vSwitch demonstrated a 5% advantage for throughput and latency while achieving a slightly lower improvement for CPU consumption. Open vSwitch is recommended for this payload sizes.

KVM guests and `uperf` pairs running on the same KVM host using a large MTU size

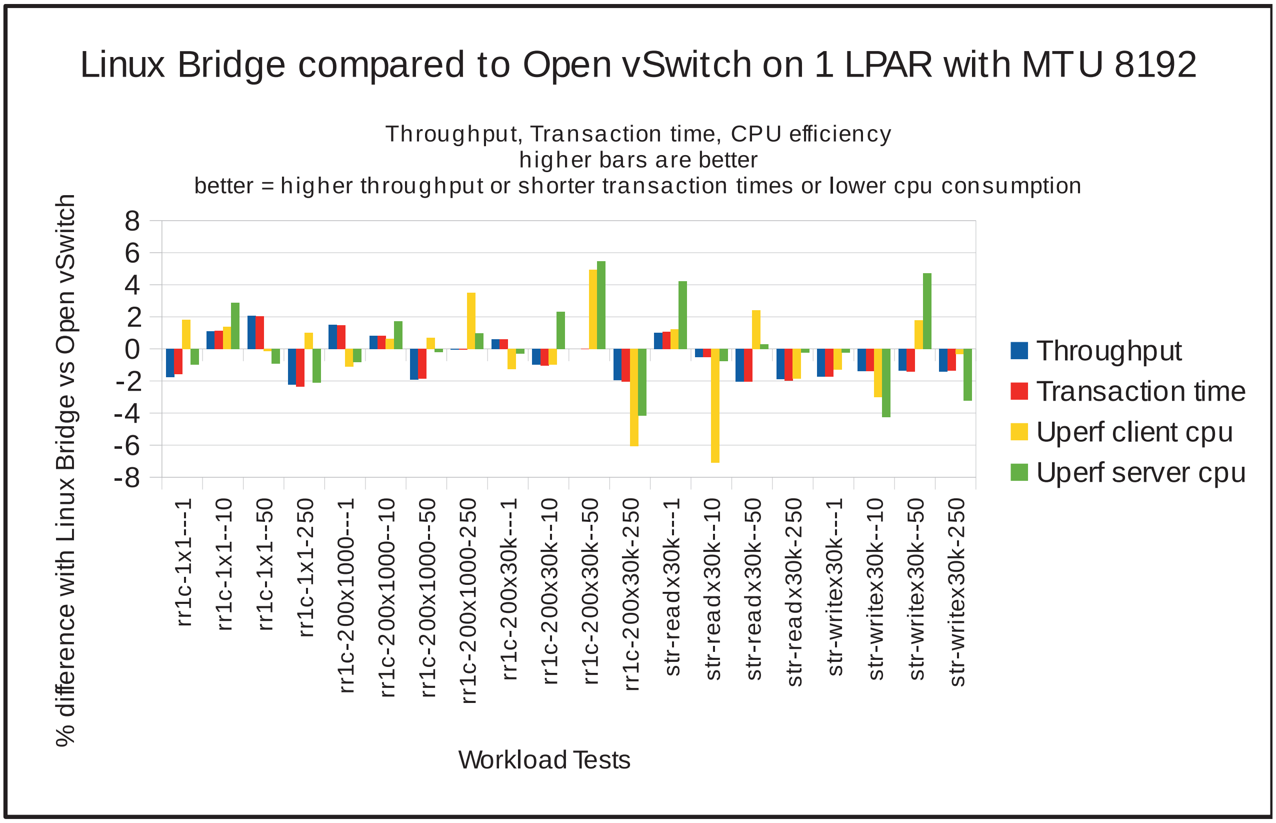


Figure 15. Single LPAR Linux Bridge compared to Open vSwitch with large MTU size

Transactional and streaming performance observations:

- Behavior is nearly identical, with a trend to a very minor drop in performance for streaming workloads.

Conclusion:

- Linux bridge results are similar to Open vSwitch across all the tests. For this reason, either would be an equally acceptable choice when using a bridge is desirable.

KVM guests and `uperf` pairs running across separate KVM hosts using a small MTU size

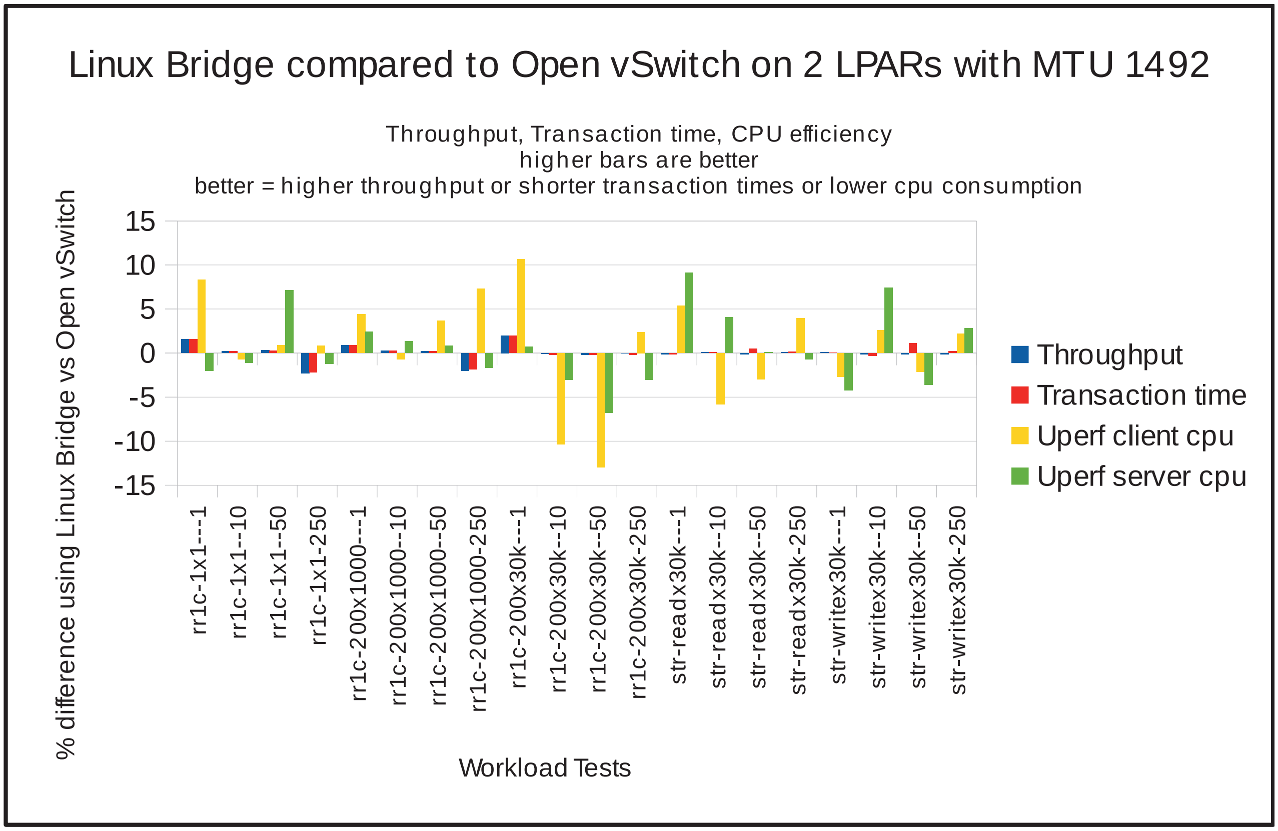


Figure 16. Linux Bridge compared to Open vSwitch across multiple LPARs with a small MTU

Transactional and streaming performance observations:

- The behavior is essentially equivalent across all workload tests.
- The difference in CPU efficiency are mixed. As seen in previous graphs, the `uperf` client and server CPU efficiencies tend to vary +/- a fair amount. With no clear trend visible across these runs, the variations are therefore considered inconsequential for comparison purposes.

KVM guests and `uperf` pairs running across separate KVM hosts using a large MTU size

Performance observations:

- The results of the large MTU size are almost identical to the small MTU results, and have therefore been omitted.

Overall conclusions when comparing a Linux bridge to Open vSwitch in our tests

For single LPAR using a small MTU size:

- Open vSwitch is recommended over the Linux bridge. The Linux bridge falls behind Open vSwitch across most tests for throughput and transaction time (latency), where the most notable difference was observed with large transactional payloads.

For single LPAR with large MTU size or multiple LPARs with either MTU size:

- The differences between both bridges is negligible as they delivered nearly equivalent performance characteristics. In this case, aspects other than performance might influence your choice of bridge.

Other considerations

- Like Open vSwitch, the Linux bridge does not require any special hardware support (ie. no switch with *hairpin* mode required) to enable the KVM host and KVM guests to communicate directly.
- The Linux bridge can be configured to provide a KVM host isolation mode. Unlike MacVTap, a Linux bridge does not require being attached to a KVM host interface in order to operate, providing a pure virtual and isolated network.
- By only connecting KVM guests to a Linux bridge and not connecting the bridge to the external facing host interface, the KVM guests can communicate with each other and the KVM host while being detached and isolated from all network traffic originating from or destined to go outside of the KVM host environment.
- Has been part of standard Linux installs for years. Requires no additional packages to install or learn.

Chapter 7. Network Performance Tuning

The process of determining the optimizations in this paper was in part drawn from past experience and from analysis of data captured during the execution of `uperf`.

This topic covers the operating system settings that will directly affect the networking performance in Linux and KVM guests. Here is the list of all the settings that were changed from the operating system default settings.

Summary of used /proc settings

The following table is a list of all the settings (located in the **/proc** file system) that were used/adjusted to obtain the results contained in this paper.

Sysctl Variable	Sysctl value
kernel.randomize_va_space	0
net.core.netdev_max_backlog	25000
net.core.rmem_max	4136960
net.core.wmem_max	4136960
net.ipv4.tcp_congestion_control	cubic
net.ipv4.tcp_fin_timeout	1
net.ipv4.tcp_limit_output_bytes	131072
net.ipv4.tcp_low_latency	0
net.ipv4.tcp_max_tw_buckets	450000
net.ipv4.tcp_rmem	4096 87380 4136960
net.ipv4.tcp_tw_reuse	1
net.ipv4.tcp_wmem	4096 16384 4136960

Using Sysctl to override default settings

Rather than modifying system variables by **echo**-ing values in to the **/proc** file system directly,

```
[root@kvm(host|guest) ~] # echo "value" > /proc/sys/location/variable
```

the **sysctl** command is available to change system/network settings. **Sysctl** provides methods of overriding default setting values on a temporary basis for evaluation purposes as well as changing values permanently that persist across system restarts.

To display a list of all available sysctl variables, use the following command:

```
[root@kvm(host|guest) ~] # sysctl -a | less
```

To only list specific variables use:

```
[root@kvm(host|gFor ID uest) ~] # sysctl variable1 [variable2] [...]
```

To change a value temporarily use the **sysctl** command with the **-w** option:

```
[root@kvm(host|guest) ~] # sysctl -w variable=value
```

To override the value persistently, the **/etc/sysctl.conf** file must be changed. This is the recommend method. Edit the **/etc/sysctl.conf** file:

```
[root@kvm(host|guest) ~] # vi /etc/sysctl.conf
```

Then add/change the value of the variable:

```
[root@kvm(host|guest) ~] # variable = value
```

Save the changes and close the file. Then use the **-p** option of the **sysctl** command to load the updated sysctl.conf settings:

```
[root@kvm(host|guest) ~] # sysctl -p or sysctl -p /etc/sysctl.conf
```

The updated sysctl.conf values will now be applied when the system restarts.

General system settings

The following topics describe the general operating system settings used for the testing.

randomize_va_space

This setting controls Address Space Layout Randomization (ASLR). ASLR is a security features that can help protect against certain types of buffer overflow attacks by randomizing the base address of code, stack, heap and other program elements to prevent attacks at known, predicted or assumed locations. This feature has been available since kernel 2.6.12 back in 2005.

ASLR is configurable and the following values are supported:

0 No randomization. Everything is static.

1 Conservative randomization. Shared libraries, stack, mmap(), VDSO and heap are randomized.

2 Full randomization. In addition to elements listed in the previous point, memory managed through brk() is also randomized.

The default setting adopted by most Linux distributions is “full randomization” (2). Changing the setting to “no randomization” (0) permits the greatest chance of sharing and enhanced cache efficiency. This setting was changed to reduce the variations of the measurement results in a controlled lab environment. Users should consider the security implications when changing this setting in their environments.

Note: It is not recommended to change this parameter in a production environment.

Kernel Same-page Merging

Another action that was taken was to disable Kernel Same-page Merging (KSM). Based on prior experience where processor utilization is a critical resource and in the absence of a memory over-commitment scenario, allowing KSM to run serves to introduce overhead in the form of breaking down Transparent Huge Pages (THP) and consuming CPU cycles when scanning memory. Since there is no need for memory over-commitment for the purposes of this paper, the service which runs KSM was disabled, which can be done by executing the following commands:

```
[root@kvmhost ~] # chkconfig ksm off
[root@kvmhost ~] # service ksm stop
Stopping ksm: [ OK ]
```

This setting was changed to reduce the variations of the measurement results in a controlled lab environment.

Note: It is not recommended to change this parameter in a production environment.

Receive Packet Steering

Modern system configurations rely heavily on network connectivity for many functions. Additionally, newer network components are getting increasingly faster and faster. This results in an ever increasing load on the OS and subsystems.

While newer processors increase capacity, much of these gains are due to additional cores rather than more powerful cores. In order to keep pace with the growing network requirements and load, many newer network adapters support multiple send and receive queues and can use multiple cores to process these queues concurrently.

For adapters that do not support multiple send / receive queues, there remains a desire and need to distribute large/high network loads across multiple processor cores. To address this, the feature *Receive Packet Steering* (RPS) was developed.

RPS uses a hash algorithm, based on packet IP addresses and ports, to distribute received network traffic across multiple cores. The hash ensures packets for the same data stream are processed on the same CPU.

RPS is specified and configured separately for each network device. Each device has a setting in sysfs. The location of the settings is:

```
/sys/class/net/<device/queues/rx-<queue#>/rps_cpus
```

where:

<device>
specifies the actual name of the interface device

rx-<queue#>
represents the network queue number being set

and

An example output of this value might be:

```
[root@kvmhost ~] # cat /sys/class/net/eth0/queues/rx-0/rps_cpus
0000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000
```

The setting is comma-delimited bitmap of CPUs. The CPU numbers are from 0 to maxcpus-1. CPU 0 is the rightmost or low order bit. The default value of all bits being 0 (off) means to RPS being disabled. Each number in the bitmap is a hex value and specifies four CPU bit locations. To enable RPS for 1 or more CPUs, the individual bitmask for the selected CPUs must be set to 1. For example, if CPUs 0-3 are to be enabled for RPS, the following invocation would be used:

```
[root@kvmhost ~] # echo 0xf > /sys/class/net/eth0/queus/rx-0/rps_cpus
```

Display the resulting bitmap:

```
[root@kvmhost ~] # cat /sys/class/net/eth0/queues/rx-0/rps_cpus
0000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,0000000F
```

The benefits realized from setting RPS bitmap will vary depending on workload characteristics. Based on measurements conducted on our KVM test systems, setting the bitmap value corresponding to the total number of CPUs configured for the KVM host has obtained the best overall improvement.

This setting is recommended for the KVM host only.

Sharing an OSA adapter across multiple LPARs

IBM Z systems are designed to share many system resources across multiple LPARs configured on the system. This is true for CPUs, memory, IO and other adapters.

The tests conducted for this paper have shown that while an OSA adapter is easily shared across several LPARs, when multiple LPARs are driving extremely high loads using a shared OSA adapter, that the peak network performance is less than when each LPAR is using a separate OSA adapter.

To obtain the highest level of network performance across multiple LPARs running KVM guests on a single system, it is recommended that each KVM host LPAR be configured to use a separate OSA adapter.

Network stack settings

This topic lists the recommended tunings for the network subsystem settings.

net.core.netdev_max_backlog

This parameter sets the maximum size of the network interface's receive queue. The queue is used to store received frames after removing them from the network adapter's ring buffer. High speed adapters should use a high value to prevent the queue from becoming full and dropping packets causing retransmits. The default value of `netdev_max_backlog` is typically 1000 frames.

For the OSA adapters used in IBM Z platforms, a value of 25000 works well for most workloads.

net.core.rmem_max / net.core.wmem_max

Increase TCP read/write buffers to enable scaling to a larger window size. Larger windows increase the amount of data to be transferred before an acknowledgement (ACK) is required. This reduces overall latencies and results in increased throughput.

This setting is typically set to a very conservative value of 262,144 bytes. It is recommended this value be set as large as the kernel allows. The value used in here was 4,136,960 bytes. However, 4.x kernels accept values over 16MB.

TCPIP IPv4 settings

This topic lists all the adjustments that were made to the IPV4 settings.

net.ipv4.tcp_congestion_control

"**Network congestion** in data networking [...]" "is the reduced quality of service that occurs when a network node is carrying more data than it can handle. Typical effects include queueing delay, packet loss or the blocking of new connections." "Networks use congestion control and congestion avoidance techniques to try to avoid congestion collapse."[11]

TCP supports a number of network congestion-avoidance algorithms, each in a separate loadable module. Most Linux distribution default to using the Reno algorithms. A list of modules available to your Linux installation can be obtained with the following command:

```
[root@kvmhost ~] # sysctl net.ipv4.tcp_available_congestion_control
```

The default algorithm for most kernels is *reno*[12]. The recommended algorithm is *cubic*[13]. The scope of testing only evaluated the algorithms available in this Linux distribution. Other post-install module could be better suited to your specific workload environment. For a more comprehensive list of algorithms which may be available for the Linux distribution being used, see:

`https://en.wikipedia.org/wiki/TCP_congestion-avoidance_algorithm#Algorithms`

net.ipv4.tcp_fin_timeout

This parameter determines the length of time an orphaned (unreferenced) connection will wait before it is aborted at the local end. This parameter is especially helpful for when something happens to the remote peer which prevents or excessively delays a response. Since each socket used for connections consumes approximately 1.5K bytes of memory, the kernel must pro-actively abort and purge dead or stale resources.

The default value for this parameter is typically 60 (seconds).

```
[root@kvmhost ~] # sysctl net.ipv4.tcp_fin_timeout net.ipv4.tcp_fin_timeout = 60
```

[11] Network Congestion: Wikipedia

[12] TCP Tahoe and Reno: TCP Congestion Avoidance: Wikipedia

[13] Cubic TCP: TCP Cogestion Avoidance: Wikipedia

For workloads or systems that generate or support high levels of network traffic, it can be advantageous to more aggressively reclaim dead or stale resources. For these configurations, it is recommended to reduce this value to below 10 (seconds).

net.ipv4.tcp_limit_output_bytes

Using this parameter, TCP controls small queue limits on per TCP socket basis. TCP tends to increase the data in-flight until loss notifications are received. With aspects of TCP send auto-tuning, large amounts of data might get queued at the device on the local machine, which can adversely impact the latency for other streams. `tcp_limit_output_bytes` limits the number of bytes on a device to reduce the latency effects caused by a larger queue size.

The default value is 262,144 bytes. For workloads or environments where latency is higher priority than throughput, lowering this value can improve latency. For these tests, this value was set to 131,072 bytes.

net.ipv4.tcp_low_latency

The normal TCP stack behavior is set to favor decisions that maximize network throughput. This parameter, when set, tells TCP to instead make decisions that would prefer lower latency.

The default value is 0 (off). For workloads or environments where latency is a higher priority, the recommended value is 1 (on).

net.ipv4.tcp_max_tw_buckets

Specifies the maximum number of sockets in the “time-wait” state allowed to exist at any time. If the maximum value is exceeded, sockets in the “time-wait” state are immediately destroyed and a warning is displayed. This setting exists to thwart certain types of “Denial of Service” attacks. Care should be exercised before lowering this value. When changed, its value should be increased, especially when more memory has been added to the system or when the network demands are high and environment is less exposed to external threats.

The default value is 262,144. When network demands are high and the environment is less exposed to external threats the value can be increased to 450,000.

net.ipv4.tcp_rmem

Contains three values that represent the minimum, default and maximum size of the TCP socket receive buffer.

The minimum represents the smallest receive buffer size guaranteed, even under memory pressure. The minimum value defaults to 1 page or 4096 bytes.

The default value represents the initial size of a TCP sockets receive buffer. This value supersedes `net.core.rmem_default` used by other protocols. The default value for this setting is 87380 bytes. It also sets the `tcp_adv_win_scale` and initializes the TCP window size to 65535 bytes.

The maximum represents the largest receive buffer size automatically selected for TCP sockets. This value does not override `net.core.rmem_max`. The default value for this setting is somewhere between 87380 bytes and 6M bytes based on the amount of memory in the system.

The recommendation is to use the maximum value of 16M bytes or higher (kernel level dependent) especially for 10 Gigabit adapters.

net.ipv4.tcp_tw_reuse

Permits sockets in the “time-wait” state to be reused for new connections.

In high traffic environments, sockets are created and destroyed at very high rates. This parameter, when set, allows “no longer needed” and “about to be destroyed” sockets to be used for new connections. When enabled, this parameter can bypass the allocation and initialization overhead normally associated with socket creation saving CPU cycles, system load and time.

The default value is 0 (off). The recommended value is 1 (on).

Note: Consult with your technical expert to ensure this change is valid in your configuration.

net.ipv4.tcp_wmem

Similar to the `net.ipv4.tcp_rmem` this parameter consists of 3 values, a minimum, default, and maximum.

The minimum represents the smallest receive buffer size a newly created socket is entitled to as part of its creation. The minimum value defaults to 1 page or 4096 bytes.

The default value represents the initial size of a TCP sockets receive buffer. This value supersedes `net.core.rmem_default` used by other protocols. It is typically set lower than `net.core.wmem_default`. The default value for this setting is 16K bytes.

The maximum represents the largest receive buffer size for auto-tuned send buffers for TCP sockets. This value does not override `net.core.rmem_max`. The default value for this setting is somewhere between 64K bytes and 4M bytes based on the amount of memory available in the system.

The recommendation is to use the maximum value of 16M bytes or higher (kernel level dependent) especially for 10 Gigabit adapters.

Network interface settings

This topic lists the changes that were made to all the network interfaces used for the network measurements.

This includes all the interfaces used in the KVM guests as well as the KVM guest interfaces in the KVM hosts (that is, `Macvtap#` or `vnet#`) as well as the KVM host OSA interfaces including any configured software bridges for settings that apply.

Maximum Transmission Unit

“Maximum transmission unit (MTU) is the maximum size (in bytes) of one packet of data that can be transferred in a network. All hops (a portion of a signal's journey from source to receiver) that are a part of the communication must be able to receive and transmit at the same MTU. However, if one of the hops in the network path is using a lower MTU size than the originating host, the package is retransmitted in a smaller size.

The default MTU size for Ethernet devices is 1500 bytes. In a distributed environment it is necessary to configure all Linux servers and switches/routers to communicate at the same MTU size to take advantage of it. In a Linux on System z® environment, it is a relatively low effort to set up all internal communication using 8992 bytes of MTU size.

By adjusting the MTU sizes, you can improve the performance of your application and database servers.

For example, typically, the larger the packet size, the better the performance because fewer packets are needed to communicate between the driver and the database. Fewer packets means fewer network round trips to and from the application. Fewer packets also require less disassembly and reassembly, and ultimately, use less CPU.[14]”1

The testing performed here compares a normal and large MTU size. The normal or default MTU size typically used is 1500 bytes and for a larger MTU size 9000 bytes tends to be the common choice. While 9000 bytes is typically used as a large MTU size, it also spans more than two physical 4K pages of memory. The larger MTU size tested here was adjusted down to fit exactly in to 2 4K pages which equals 8192 bytes.

The degree of improvement varies depending on whether the KVM guests are running on a single KVM host or on separate KVM hosts.

[14] Set up Linux on IBM System z for Production

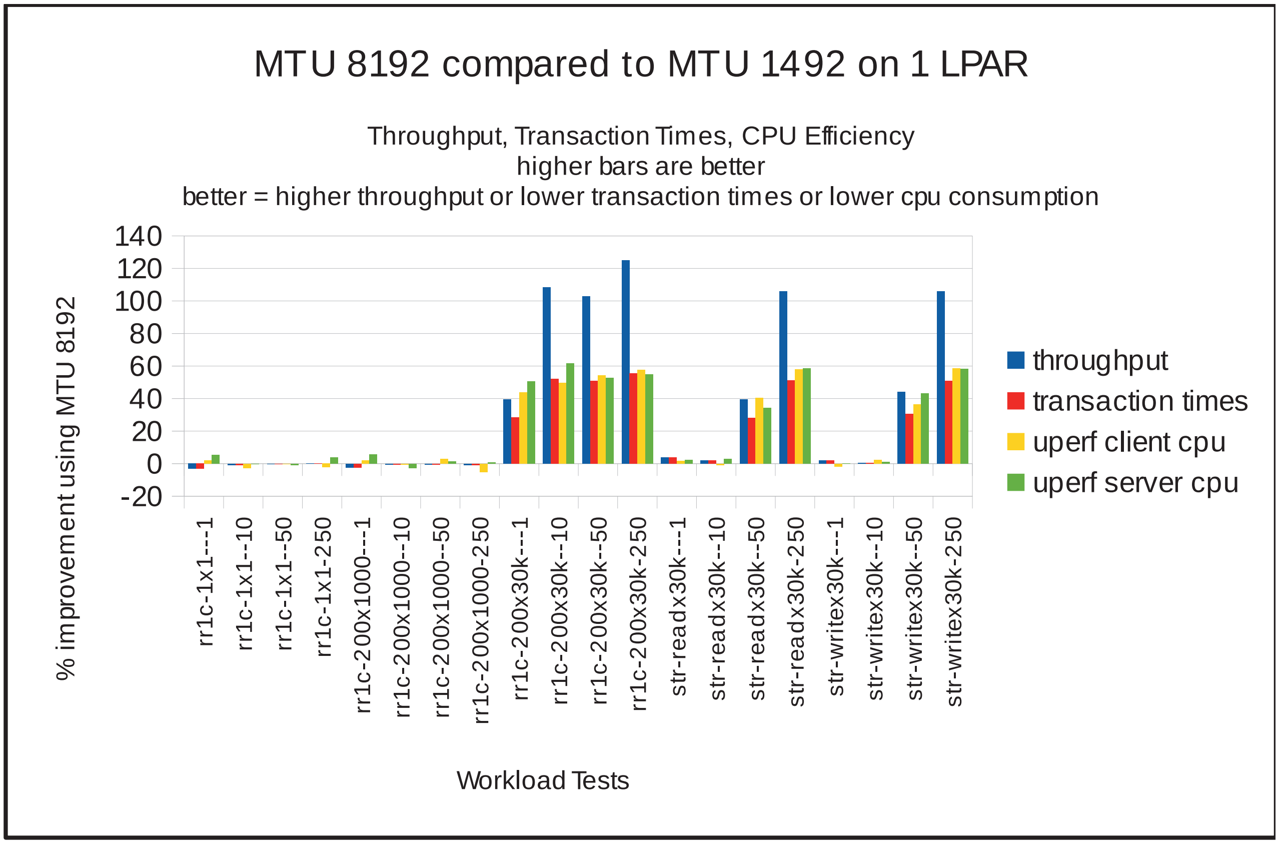


Figure 17. Throughput, Latency and CPU efficiency using a large MTU size vs the default MTU size with KVM guests running on a single KVM host LPAR

Figure 17 shows the improvements when a large MTU size is used between KVM guests running on the same KVM host. In this configuration, improvements are seen when the datagram or payload size is larger than the default MTU size. Three workload test types have payload sizes of 30K bytes. Each of these workload test types obtain substantial (up to 120%) improvements in throughput, latency and CPU efficiency when the larger MTU size is used.

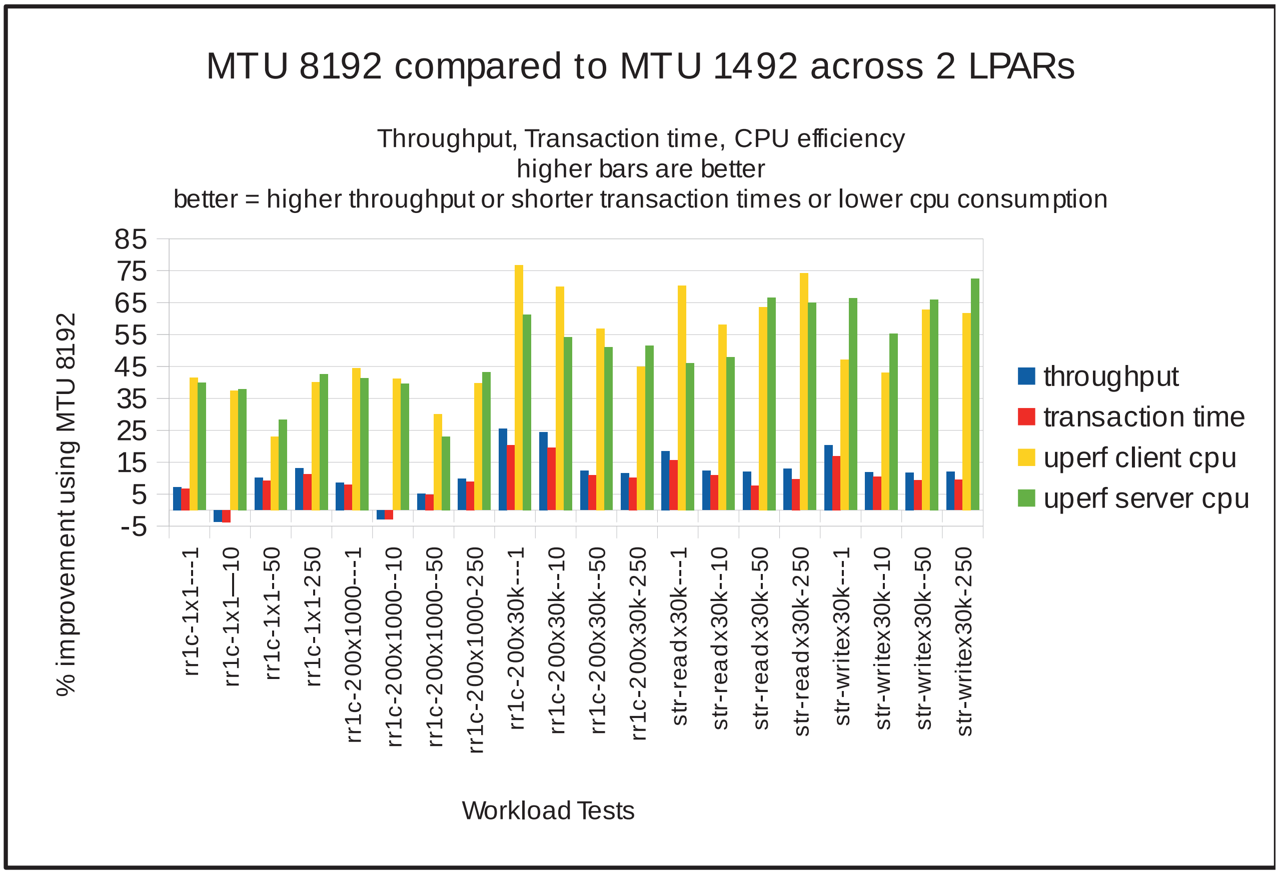


Figure 18. Throughput, latency and CPU consumption using normal and large MTU sizes with KVM guests running on separate LPARs

In Figure 18 we can see the improvements of a larger MTU size when KVM guests are running on separate KVM hosts. In this configuration, the maximum throughput is limited to the speed of the physical network fabric. Here the larger MTU size provides 5-20% improvement in throughput and transaction time in all but 2 workload test types. From a CPU perspective the efficiency improved by 25-75% across all test types.

These results clearly indicate that using a larger MTU size provides improvements in almost all use case scenarios and is the recommendation to use to configure your environments. However, to take advantage of the potential gains, the entire network path must be configured to use the larger MTU size. This may not always be possible based on external factors such as hardware limitations or restrictions enforced by other shared network infrastructure systems and components.

The MTU size can be set dynamically using the command:

```
[root@kvm(host|guest) ~] # ip link set dev {interface} mtu NEWMTUSIZE
```

However this method of setting the MTU size is not persistent across reboots.

To make the change persistent it is recommended to include the **MTU=** paremeter in the **“ifcfg-{interface}”** file for the target interface.

MTU=8192

After making this change, the interface must be restarted for the change to take effect. To stop and start the interface use these commands:

```
[root@kvm(host|guest) ~] # ifdown {interface}; ifup {interface}
```

Note: The MTU size will typically default to the smallest MTU size in the transmission path between source and target endpoints. When running KVM guests and communicating external to host, the host may or may not be an active participant of the communication path MTU size determination, depending on the network model used by the guest.

Note: When using MacVTap which uses a direct connection to a host interface, the host is not an active participant in the transmission path's MTU size determination. In this case, if the MTU size in the host is smaller than the MTU size of the KVM guest and the KVM guest attempts to send packets that are larger than the MTU size in the host, those packets will stall in the host. The host will not fragment the larger packets down to fit in to its MTU and the connection will hang until it times out. It is therefore necessary to set the host MTU size equal to or greater than the MTU size set in the KVM guests.

buffer_count

The buffer count parameter is a network parameter for QDIO devices for [Linux on IBM Z]. This parameter allows Linux servers to receive more network packets in order to increase throughput. The default buffer count value for [Linux on IBM Z] is 16 . This parameter must be defined for each network device and allocates 1 MB of memory. A buffer count of 128 leads to 8 MB of memory consumption.

You can check the actual buffer count by using the **`lsqeth -p <interface>`** command.

The configuration of the buffer count must be done with the virtual interface in an offline state.[15]

It is recommended to add this setting in “ifcfg-{interface}” file for the target interface. The OPTIONS= parameter needs to be added or edited to include “buffer_count=128”:

```
OPTIONS="buffer_count=128"
```

The interface must be restarted for the change to take effect.

Network checksumming

The OSA network cards support two checksumming options. The first option is to use software checksumming and the second option is to perform checksumming in the hardware (on the OSA card).

To reduce the load from the CPUs and perform checksumming faster, the checksumming setting was changed from the default of software checksumming to hardware checksumming.

The option is changed in “ifcfg-{interface}” file for the target interface. The OPTIONS= parameter should be added or edited to include “checksumming=hw_checksumming”.

[15] Set up Linux on IBM System z for Production

```
[root@kvm(host|guest) ~] # OPTIONS="checksumming=hw_checksumming"
```

After making this change, the interface must be restarted for the change to take effect.

Transmit (TX) Queue length

"There is a setting available to adjust the size of the queue between the kernel network subsystems and the driver for network interface card. Like with other buffers, it is recommended to appropriate set the queue size to prevent losses resulting from buffer overflows. Therefore careful tuning is required to ensure that the sizes of the queues are optimal for your network connection."

"These settings are especially important for TCP as losses on local queues will cause TCP to fall into congestion control – which will limit the TCP sending rates. Meanwhile, full queues will cause packet losses when transporting udp packets."

"There are two queues to consider, the netdev_backlog (see 'net.core.netdev_max_backlog' in "Network stack settings" on page 46) which relates to the receive queue size and **txqueuelen** which determines the transmit queue size."[16]

The default transmit queue size (**txqueuelen** setting) for a network device on IBM Z is 1000. This value is adequate for Gigabit network devices. However, for devices with 10 Gbs or greater, the **txqueuelen** setting should be increased to avoid overflows that drop packets.

Similarly, choosing a value that is too large can cause added overhead resulting in higher network latencies.

To query the transmit queue size of the device on your system use either of the following:

```
[root@kvm(host|guest) ~] # ifconfig <interface-name>
eth0 Link encap:Ethernet HWaddr e4:1f:13:ba:c7:04
     inet addr:9.53.92.168 Bcast:9.53.92.255 Mask:255.255.255.0
     inet6 addr: fe80::e61f:13ff:feba:c704/64 Scope:Link
     UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
     RX packets:165275988 errors:0 dropped:0 overruns:0 frame:0
     TX packets:169557966 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:1000
     RX bytes:87361147022 (87.3 GB) TX bytes:117748544954 (117.7 GB)
```

The transmit queue size is highlighted.

or

```
[root@kvm(host|guest) ~] # ip link show dev <interface-name>
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group
default qlen 1000
link/ether e4:1f:13:ba:c7:04 brd ff:ff:ff:ff:ff:ff
```

The reported transmit queue size value is **highlighted** in both examples.

[16] http://datatag.web.cern.ch/datatag/howto/tcp.html

To change the default **txqueuelen** value, either of the following will work:

```
[root@kvm(host|guest) ~] # ifconfig <interface-name> txqueuelen <new-value>
```

or

```
[root@kvm(host|guest) ~]# ip link set txqueuelen <new-value> dev <interface-name>
```

To determine the optimal setting for your environment and workload some experimentation may be required. The **txqueuelen** value that produced the best results for our tests was 2500. This value is good starting point.

KVM settings

This topic describes the KVM settings that were used for this study.

halt_poll_ns

On newer kernel releases there is a KVM module parameter `/sys/module/kvm/parameters/halt_poll_ns` that can be used to alter the behavior of how idle KVM guest virtual CPUs (vcpus) are handled.

When a vcpu in a KVM guest has no work to do (no threads waiting to run), QEMU traditionally halts the idle vcpus. This setting specifies a period of time (in nanoseconds) that a vcpu will wait and poll looking for new work *before* the vcpu is halted.

When new work arrives during the polling period (before the vcpu is halted), the vcpu is immediately ready to execute the work. If the vcpu has been halted when new work arrives, the vcpu must be brought out of the halt state before the new work can be started. The time it takes to transition the vcpu from halted to running state induces additional latency which negatively impacts latency sensitive workloads.

The introduction of the polling period can improve the responsiveness of small payload transactional network workloads.

Specifying a poll time in general is expected to have a potentially negative impact to CPU utilization. Instead of halting a vcpu immediately, the vcpu is now polling (looping) while it waits for new work. The additional poll time can increase the CPU utilization.

So this setting attempts to balance the trade-off of potentially increased CPU consumption in order to provide better responsiveness and lower latencies to improve throughput and operations/sec. The amount of polling time that is ideal for a particular network workload can vary. Users can experiment with different values to find whats optimal for their workloads.

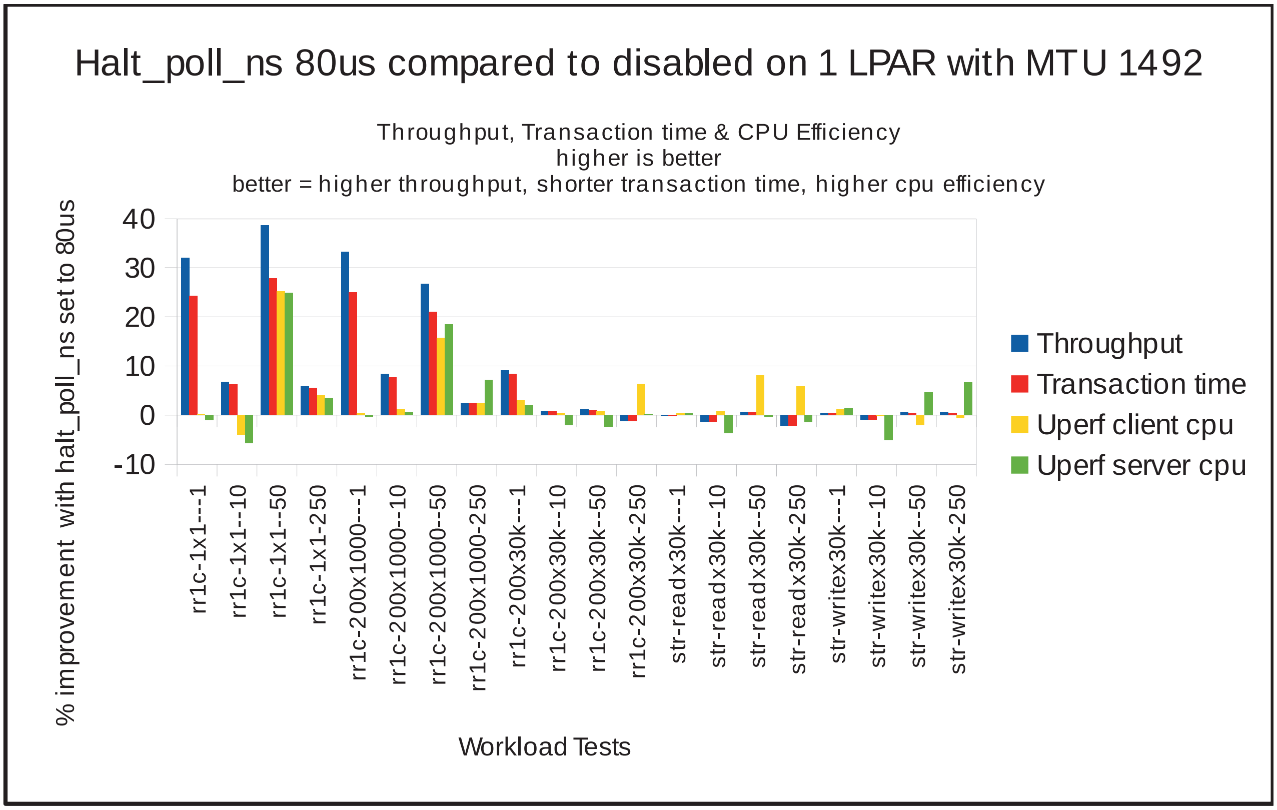


Figure 19. halt_poll_ns improvements to transactional workloads

Experiments were conducted using the `uperf` workloads defined earlier. For these workload tests, a `halt_poll_ns` value of *80,000 ns* produced the best overall results. In Figure 19, we see a number of the small payload transactional tests get up to 35% improvement with most seeing little or no negative impact to CPU efficiency.

To see what poll time value is set use the following command:

```
[root@kvmhost ~] # cat /sys/module/kvm/parameters/halt_poll_ns
```

A value of 0 means `halt_poll_ns` is disabled.

To enable or change the polling time, specify a non-zero value using the command:

```
# echo 80000 > /sys/module/kvm/parameters/halt_poll_ns
```

Changes to this setting are immediately available to running KVM guests with no restarts required.

In a environment where CPU resources are highly over-committed, this setting might be counterproductive.

Chapter 8. Summary

KVM provides open source virtualization for IBM Z and the LinuxONE platforms.

Using the combination of KVM virtualization and IBM Z and LinuxONE, a wide variety of network centric workloads are supported by IBM Z hardware. With an understanding of the network characteristics important to your environment, and knowledge of the KVM and Linux environment, you can optimize a system more quickly to achieve better performance while reducing the resource consumption leading to better efficiency and system utilization. This paper highlights considerations to help you achieve these goals, to better reach your requirements and to more fully exploit the potential of your IBM Z machines.

The table below summarizes, from the data presented earlier, the network configuration that achieved better overall performance characteristics:

LPAR/MTU configuration	Small transactional workload characteristics	Large transactional workload characteristics	Streaming workload characteristics
1 LPAR / small MTU	MacVTap	MacVTap	MacVTap
1 LPAR / large MTU	MacVTap	MacVTap	MacVTap
2 LPARs / small MTU	MacVTap better at higher user loads	MacVTap marginally better	Open vSwitch used less CPU
2 LPARs / large MTU	MacVTap better at higher user loads	Open vSwitch better at higher user loads	Open vSwitch used less CPU at higher user loads

Note: For single LPAR with large MTU size or multiple LPARs with either MTU size, the differences between Open vSwitch and the Linux bridge are negligible as they delivered nearly equivalent performance characteristics. In this case, aspects other than performance might influence your choice of bridge.

References

View a list of documents referenced in this white paper.

1. https://en.wikipedia.org/wiki/IBM_Z
2. https://www.ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_kvm_base.html
3. https://en.wikipedia.org/wiki/Logical_partition
4. www.redbooks.ibm.com/abstracts/sg248137.html
5. http://virt.kernelnewbies.org/MacVTap
6. http://www.uperf.org/
7. https://en.wikipedia.org/wiki/Private_network
8. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s2-networkscripts-interfaces_network-bridge.html
9. http://openvswitch.org/
10. https://en.wikipedia.org/wiki/Network_congestion#Avoidance
11. https://en.wikipedia.org/wiki/TCP_congestion-avoidance_algorithm#TCP_Tahoe_and_Reno
12. https://en.wikipedia.org/wiki/CUBIC_TCP
13. http://datatag.web.cern.ch/datatag/howto/tcp.html

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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 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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information in softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., 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 http://www.ibm.com/legal/copytrade.shtml

Adobe is either registered trademarks or trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of the manufacturer.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of the manufacturer.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any data, software or other intellectual property contained therein.

The manufacturer reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by the manufacturer, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

THE MANUFACTURER MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THESE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.