Linux on IBM Z

KVM Network Performance

Best Practices and Tuning Recommendations

Dr. Juergen Doelle

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries.

- A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/us/en/copytrade.shtml

The following are trademarks or registered trademarks of other companies.

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- SUSE is a registered trademark of Novell, Inc. in the United States and other countries.
- Red Hat, Red Hat Enterprise Linux, the Shadowman logo and JBoss are registered trademarks of Red Hat, Inc. in the U.S. and other countries.
- Oracle and Java are registered trademarks of Oracle and/or its affiliates in the United States, other countries, or both.

Other product and service names might be trademarks of IBM or other companies.

Introduction

- **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 support the use of KVM to create and run multiple virtual machines or guests in each LPAR.

- **Using the combination of KVM virtualization and IBM Z and IBM LinuxONE,**
  - Provides 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 virtualized systems to run various workloads, sharing resources, and improving service levels to meet demand

- **There are many possibilities to implement a virtualized network infrastructure to connect KVM guests with the outside world. This presentation**
  - Shows the various possibilities and how to set them up
  - Discusses pros and cons
  - Provides network tuning hints

- **Note:**
  - All terms like bridges, Open vSwitch etc are used in the context of libvirt and KVM

Agenda

- **Setup overview**
- **Virtual network setups**
  - MacVTap
  - Linux bridge
  - Open vSwitch
- **Network traffic forwarding**
  - NAT-based networking
  - Promiscuous mode
  - MAC address registration
  - VNIC characteristics
- **Network setup - Pros and cons**
  - Network workload description
  - MacVTap considerations
  - Open vSwitch considerations
  - Linux Bridge considerations
- **Network performance tuning**
  - Sysctl settings
  - Offloads, TX queue
  - Receive Packet Steering (RPS)

Network configurations

- **There are three virtual network device options for KVM guests**
  - MacVTap
  - Linux Bridge
  - Open vSwitch
- **Modes connecting the guest network to the physical network interface**
  I. OSA card in promiscuous mode
  II. NAT
  III. Register the guest interface at the OSA card
  New IV. OSA VNIC characteristics
  - Note:
    - One (and only one) from the options I – IV must be selected for a certain guest network interface
    - A guest can have multiple interfaces with different connection types
- **Resulting network configurations for OSA cards**

	MacVTap	Linux Bridge	Open vSwitch
OSA promiscuous mode	--	one of these	one of these
OSA Interface registration	implicitly		
OSA vnicc	--		
NAT	--		?

- **Another option is to connect the guest to the host network (no assigned physical interface)**
  - IP routing → handled in a separate presentation
    - https://www.ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_perf_latest.html#perf_latest__kvm_ip_routing

Important hints

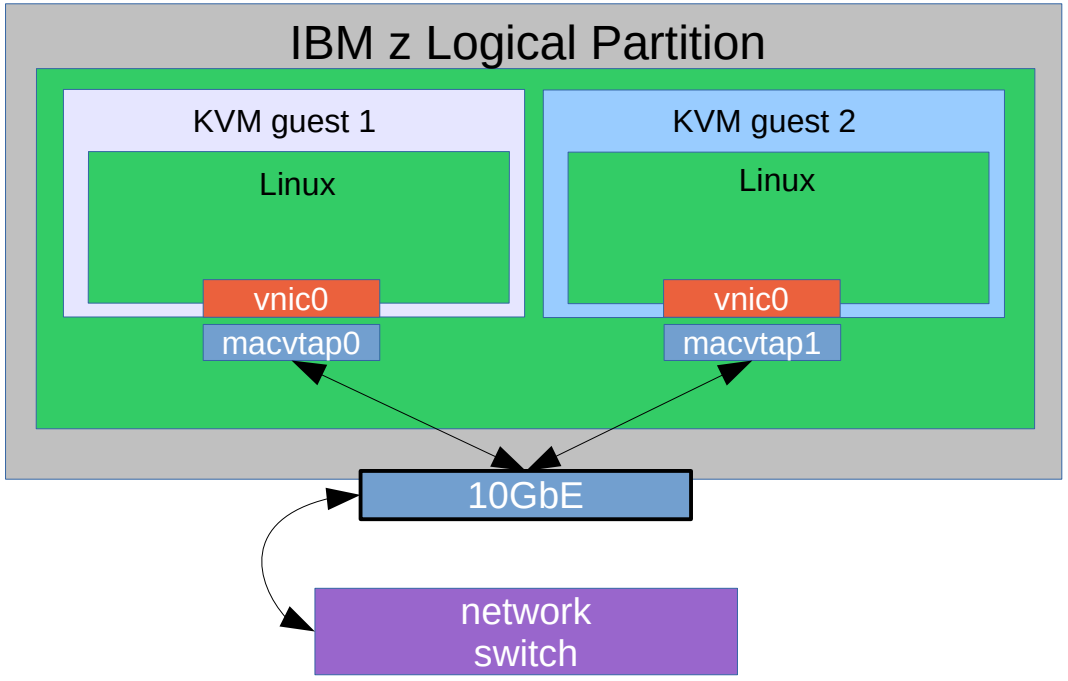
- **Note:**
  - For all scenarios: the guest interface requires its own MAC address (Layer 2)
    - It might be useful to create a schema to generate MAC addresses
  - If an OSA adapter is shared across multiple LPARs
    - Only one single LPAR can configure the OSA card for promiscuous mode at any point in time
    - Another LPAR requires a separate OSA adapter to use the promiscuous mode
    - VNIC characteristics might be an option on IBM z15

- **Recommendation: For better performance throughput and latency**
  - Use the newer vhost-net driver for KVM guests instead of the older para-virtualized virtio-net driver
    - Specify the keyword `<driver name="vhost"/>` in the guest's libvirt configuration file (domain.xml).

Agenda

- **Setup overview**
- **Virtual network setups**
  - MacVTap
  - Linux bridge
  - Open vSwitch
- **Network traffic forwarding**
  - NAT-based networking
  - Promiscuous mode
  - MAC address registration
  - VNIC characteristics
- **Network setup - Pros and cons**
  - Network workload description
  - MacVTap considerations
  - Open vSwitch considerations
  - Linux Bridge considerations
- **Network performance tuning**
  - Sysctl settings
  - Offloads, TX queue
  - Receive Packet Steering (RPS)

MacVTap - Setup



- **MacVTap simplifies virtualized networking**
- **Connects the guest**
  - Guest network interface is directly connected to the KVM host interface
    - Shortens the code path!
  - The host can **not** communicate with the guest directly

- **Modes**
  Three modes provide different level of isolation

I. Virtual Ethernet Port Aggregator (VEPA)
- Default mode
- Data flows from one endpoint
  - through the physical network card
  - to the external switch
- If the switch supports hairpin mode (not typical!), guests can communicate via the switch

II. Bridge
- Connects all endpoints directly
- Two guests in bridge mode can communicate without the switch
- **Most useful MacVTap mode** for inter-guest communication

III. Private
- Similar like VEPA but without hairpin, guest can not communicate with any other guest on the same host

MacVTap – Connect the guest

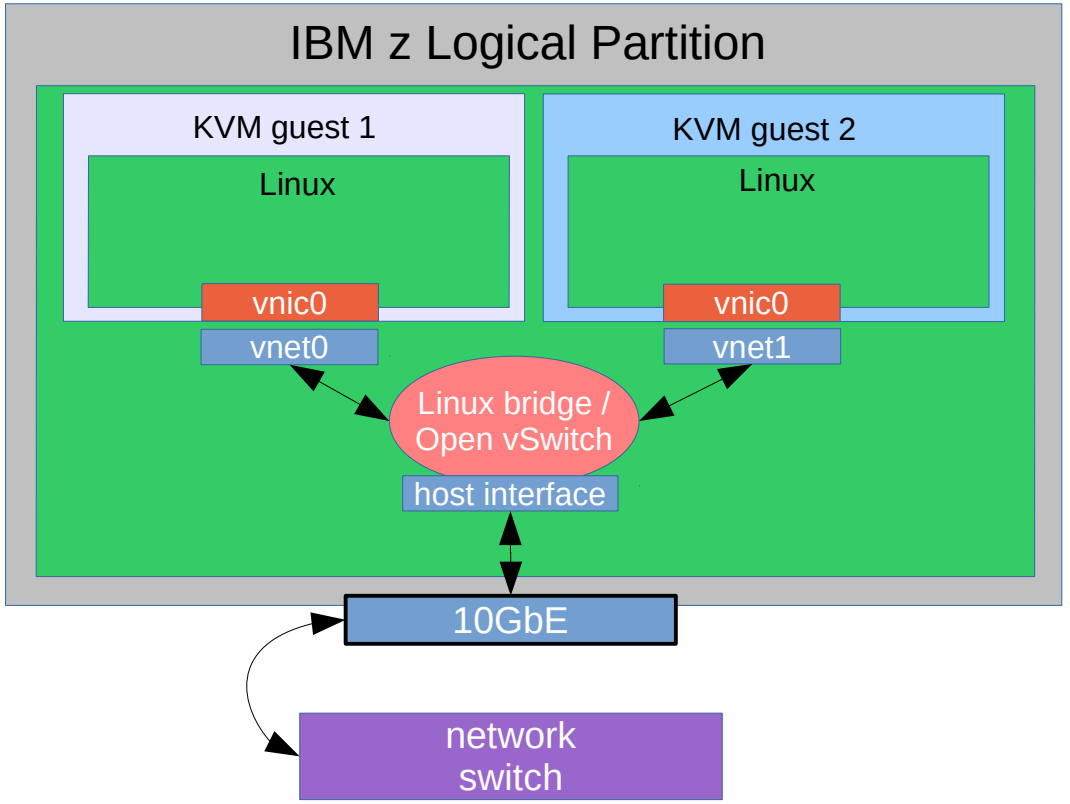
- **The driver macvlan and macvtap are loaded from libvirt when a guest is started via virsh**

- **Domain XML statements**

```
<interface type="direct">
    <source dev="eth0" mode="bridge"/>
    <mac address="12:34:56:78:9a:bc"/>
    <model type="virtio"/>
    <driver name="vhost"/>
</interface>
```

- **The relevant XML tags here are:**
  - `<interface type="direct">`
    - A direct attachment to an existing KVM host network device.
    - The host will generate a name, for example macvtap0, for this connection with the same MAC address as specified in the next statement
  - `<mac address=…`
    - Optional, when omitted a unique address is generated
    - The MAC address is automatically registered at the OSA card!
  - `<source dev="eth0" mode="bridge">`
    - Host network interface name to connect to
    - MacVTap mode

Bridges overview



- **A bridge simplifies the administration of virtualized bridged networking!**
  - The guest gets connected to the bridge, that's it
  - The connection to the hardware and network is managed on one single point, the bridge
- **Bridges provide**
  - Creation of pure virtual networks
  - The host can communicate with the guest
  - Depending on the type of bridge used very powerful features are available
- **Two types of bridge are available**
  - Linux bridge
  - Open vSwitch

Linux bridge – Setup a bridge

- **The standard Linux network bridge.**
  - A network bridge is a Link Layer device which forwards traffic between networks based on MAC addresses (Layer 2 device).
  - Management tools package bridge-utils

- **Show all current instances of the ethernet bridge**
  - `[root@kvmhost ~] # brctl show`
- **Create a Linux bridge on a KVM host**
  - `[root@kvmhost ~] # brctl addbr <bridge-name>`
- **Attach a Linux bridge to an OSA interface in the KVM host**
  - `[root@kvmhost ~] # brctl addif <bridge-name> <host-interface-name>`
- **Linux network bridge might also be configured for the NAT mode**

Linux bridge – Connect the guest

- **Add or change the KVM guest's configuration to use the Linux bridge**
  - Include the definition in the domain xml configuration file
  - ```
    <interface type="bridge">
        <source bridge="bridge-name"/>
        <mac address="11:22:33:44:55:66"/>
        <target dev="my-vnet0"/>
        <model type="virtio"/>
        <driver name="vhost"/>
    </interface>
    ```
  - This will create for the guest
    - A network interface `"my-vnet0"` in the host (ip link show)
    - Connected to the bridge `"bridge-name"`
    - Assigned to the guest
  - The MAC address field is optional, if omitted, the libvirt daemon will generate a unique value.
  - The target field is optional, if omitted, the libvirt daemon will generate a unique value, vnet0...
    ```

Open vSwitch – Setup a bridge

- **Open vSwitch (abbreviated to OVS) is a production quality, multi-layer virtual switch.**
  - It is designed to enable massive network automation through programmatic extension
  - Supporting standard management interfaces and protocols, for example
    - Channel bonding, including mode LACP (IEEE 802.1AX-2008)
    - Standard 802.1Q VLAN model with trunking
    - Per VM interface traffic policing
    - Remote configuration protocol with C and Python bindings
    - For details see http://www.openvswitch.org/features
  - Open vSwitch has many features that can not be found in the standard Linux bridge

- **To use Open vSwitch, the service must be enabled and started**
  - ```
    [root@kvmhost ~] # systemctl enable openvswitch.service
    [root@kvmhost ~] # systemctl start openvswitch.service
    ```
- **The enablement needs only be done once, it will persist across KVM host restarts.**
- **To create an Open vSwitch bridge use:**
  - ```
    [root@kvmhost ~] # ovs-vsctl add-br <bridge-name>
    ```
- **Attach an OSA device to the Open vSwitch bridge:**
  - ```
    [root@kvmhost ~] # ovs-vsctl add-port <bridge-name> <host-interface-name>
    ```
- **Move the IP address from the OSA device to the Open vSwitch**
  - ```
    ip addr del <OSA IP addr>/<scope> dev <interface-name>
    ip addr add <OSA IP addr>/<new scope> dev <bridge-name>
    ifup <bridge-name>
    ```

Open vSwitch – Connect the guest

- **Add or change the KVM guest's configuration to use the Open vSwitch bridge**
  - Include the definition to define the interface in the domain xml configuration file
  - ```
    <interface type="bridge">
        <source bridge="bridge-name"/>
        <virtualport type="openvswitch"/>
        <mac address="11:22:33:44:55:66"/>
        <target dev="my-vnet0"/>
        <model type="virtio"/>
        <driver name="vhost"/>
    </interface>
    ```
  - This will create for the guest
    - A network interface `"my-vnet0"` in the host (ip link show)
    - Connected to the bridge `"bridge-name"`
    - Assigned to the guest
  - The MAC address field is optional, if omitted, the libvirt daemon will generate a unique value.
  - The target field is optional, if omitted, the libvirt daemon will generate a unique value, vnet0...
  - The virtualport statement is the only difference to the Linux bridge
  - The UUID of the OVS bridge can be obtained

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

Open vSwitch – More information

- **For messages see logs in /var/log/openvswitch**
- **For further reading see**
  - http://docs.openvswitch.org/en/latest

  Especially
  - http://docs.openvswitch.org/en/latest/faq/configuration
  - http://docs.openvswitch.org/en/latest/faq/issues

Agenda

- **Setup overview**
- **Virtual network setups**
  - MacVTap
  - Linux bridge
  - Open vSwitch
- **Network traffic forwarding**
  - NAT-based networking
  - Promiscuous mode
  - MAC address registration
  - VNIC characteristics
- **Network setup - Pros and cons**
  - Network workload description
  - MacVTap considerations
  - Open vSwitch considerations
  - Linux Bridge considerations
- **Network performance tuning**
  - Sysctl settings
  - Offloads, TX queue
  - Receive Packet Steering (RPS)

Network traffic forwarding

- **Connect the guest to the outside world**
- **KVM guest interfaces need to run in layer 2 providing a unique MAC address**

- **MacVTap**
  - Driver registers and manages implicitly the MAC address at the OSA card (no user actions required)

- **OpenvSwitch/Linux bridge**
  - By default, the OSA card only forwards network traffic destined to devices that are known
  - There are two/three OSA interface configuration modes
    - OSA promiscuous mode → let the OSA card forward (nearly) everything
    - Register MAC addresses manually → let the OSA card know the guest devices
    - New with IBM z15: VNIC characteristics → let the OSA card learn the guest devices
  - Configuration of one of them is required!
  - It is possible to register additional MAC addresses for an OSA card running in promiscuous mode

- **Linux bridge in NAT mode**
  - Does not require that the MAC addresses from the guests are registered!

KVM NAT based networking

- **Network Address Translation (NAT) -based networking is commonly provided and enabled as default by most major Linux distributions that support KVM virtualization.**
  - To enable a guest OS to get outbound connectivity
  - Allowing KVM guests sharing the same bridge to communicate with each other
    - Even if the bridge is not connected to an interface in the KVM host (pure virtual network)

- **Characteristics of NAT interfaces**
  - Typically configured to use internally private IP addresses from a 192.168.x.x subnet
  - The guest IP address is not visible outside of the KVM host running the NAT!
    - Externally is only the IP address of the OSA card visible
    - No MAC registration of the guest on the network card required
  - Related with overhead
    - NAT behavior is normally implemented using a linux firewall that employs static and dynamic firewall rules.
    - Affects throughput and latency as well as potentially increases the consumption of CPU

KVM NAT based networking – Managing the bridge with virsh

- **List which networks have been defined to the libvirt daemon for use by KVM guests**
  - ```
    root@kvmhost ~] # virsh net-list
    Name        State       Autostart   Persistent
    ----------------------------------------------------------
    default     active      yes               yes
    ```
- **A network named “default” is often preconfigured from libvirt and connected to bridge virbr0**
  - To disable (stop): `virsh net-destroy default`
  - Prevent creation at boot: `virsh net-autostart default --disable`
  - Undefine the configuration: `virsh net-undefine default`
- **To view configuration details of a specific network defined in libvirt**
  - `virsh net-dumpxml <libvirt-network-name>`
  - ```
    virsh net-info default
    Name:           default
    UUID:           a5f33e32-cc72-4059-89ca-691faf4d4dec
    Active:         yes
    Persistent:     yes
    Autostart:      yes
    Bridge:         virbr0
    ```
  - Note: there are two terms
    - the virtual network name: *default* (only available with Linux bridges)
    - the bridge name: *virbr0*
    - either can be used in the domain.xml to define the source for the network
- **For more information to `virsh net-`... commands see** `man virsh`

KVM NAT based networking – Define a bridge running NAT

- **Define a new NAT bridge**
  - Create a XML file, e.g. new-kvm-network.xml

```
<network>
    <name>newnatnetwork</name>
    <forward mode="nat">
        <nat> <port start="1024" end="nnnnn"/> </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.100" end="192.168.X.254"/> </dhcp>
    </ip>
</network>
```

  - Change attributes <name>, <bridge name> and <ip address> to suite your needs.
  - Replace 'X' with a meaningful subnet value and adapt the dhcp address range
  - Choose different values than the “default” network, check with
    - `virsh net-dumpxml default`
  - Add the new network definition XML file to libvirt
    - `virsh net-create ~/new-nat-network.xml`
  - Set the new network to automatically startup each time the KVM host is rebooted
    - `virsh net-autostart <network-name-from-xml>`
- **Note:**
  - This is an alternative to create a bridge using the brctl command
- **Add or change the KVM guest's configuration to use this bridge**
  - `<source bridge="bridge-name"/>` or `<source network="network-name"/>`

Promiscuous mode

- **By default, the OSA card only forwards network traffic destined to devices that are known**
  - For larger network configuration with multiple LPARs the manual registration of MAC addresses becomes more and more complex.

- **The firmware of newer OSA cards supports a configuration option called promiscuous mode.**
  - Removes the requirement for OSA MAC address registration
  - Only one LPAR can enable the promiscuous mode for an OSA port!
  - Other KVM LPARs can still register the MAC addresses from their guest
    - This might not work when VLANs are involved

Promiscuous mode – Default settings

- **Display the configuration of an OSA card**
  - `lsqeth <interface-name>`
  -
```
[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
```

Promiscuous mode, cont.

- **Enable promiscuous mode on the OSA card (default none)**
  - `echo "primary" > /sys/class/net/<interface-name>/device/bridge_reflect_promisc`
- **And enable promiscuous mode in the Linux Kernel to activate it:**
  - `root@kvmhost] # ip link set dev <interface-name> promisc on`
  - `lsqeth o5s_10g_0`

```
Device name                     : o5s_10g_0
-------------------------------------------------------------------------------
        card_type               : OSD_10GIG
        . . .
        layer2                  : 1
        isolation               : none
        bridge_role             : primary
        bridge_state            : active
        bridge_hostnotify       : 0
        bridge_reflect_promisc  : primary
        switch_attrs            : unknown
        vnicc/bridge_invisible  : n/a (BridgePort)
        vnicc/flooding          : n/a (BridgePort)
        vnicc/learning          : n/a (BridgePort)
        vnicc/learning_timeout  : n/a (BridgePort)
        vnicc/mcast_flooding    : n/a (BridgePort)
        vnicc/rx_bcast          : n/a (BridgePort)
        vnicc/takeover_learning : n/a (BridgePort)
        vnicc/takeover_setvmac  : n/a (BridgePort)
```

- **With promiscuous mode active, manual registration of the guest MAC address is no longer required!**

Manual MAC address registration

- **The OSA cards maintains a “Forwarding Database” for registered MAC addresses**
- **List the Forwarding Database entries for all OSA cards**
  - ```
    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
    ```
- **List the Forwarding Database entries associated to a specific OSA device**
  - `bridge fdb show dev <interface-name>`

    ```
    [root@kvmhost] # bridge fdb show dev 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
    ```
- **Register a new device on the OSA card, use this command:**
  - `bridge fdb add <new-device-mac-address> dev <interface-name>`
    - `bridge fdb add 12:34:56:78:9a:bc dev 10gb1`
  - deregister with `bridge fdb del <new-device-mac-address> dev <interface-name>`
- **Verify the result at the OSA address table**
  - `qethqoat 10gb1`
  - If the registered MAC address does not appear under vmac, this is a hint that the MAC is already registered on another LPAR sharing this OSA card
- **see also *man bridge***
    ```

Virtual Network Interface Controller (VNIC) characteristics

- **Applies only for layer 2 devices**
  - OSA and HiperSockets
  - Requires IBM z15 (firmware bundle 28a) for OSA
  - HiperSockets devices support it with z14 GA.

- **VNIC characteristics**
  - Configure a layer 2 network device to receive all unknown traffic (like promiscuous mode)
  - Overcomes the limitation of the the promiscuous mode
    - Can be applied on multiple LPARs on the same CEC
- **A network device can be configured either in promiscuous mode or with VNIC characteristics (mutually exclusive)**

- **Always set at least the attributes flooding and learning, for example**
  - chzdev
    - `chzdev <device_bus_id> vnicc/flooding=1 vnicc/learning=1`
  - sysfs attributes in /sys/devices/qeth/<device_bus_id>/vnicc:
    - `echo 1 >/sys/devices/qeth/<device_bus_id>/vnicc/learning`
    - `echo 1 >/sys/devices/qeth/<device_bus_id>/vnicc/flooding`
  - or device udev rule
    - `ATTR{[ccwgroup/<device_bus_id>]vnicc/flooding}="1"`
      `ATTR{[ccwgroup/<device_bus_id>]vnicc/learning}="1"`

  - `<device_bus_id>` is a device number with a leading "0.<n>.", where <n> is the subchannel set ID

Virtual Network Interface Controller (VNIC) characteristics

- **verify with** `lszdev <device_bus_id> --info` **or lsqeth**

```
lszdev 0.0.e100 --info

DEVICE qeth 0.0.e100:0.0.e101:0.0.e102
---

  ATTRIBUTE                ACTIVE                          PERSISTENT
  bridge_hostnotify        "n/a (VNIC characteristics)"    -
  bridge_reflect_promisc   "n/a (VNIC characteristics)"    -
  bridge_role              "n/a (VNIC characteristics)"    -
  buffer_count             "128"                           "128"
  hw_trap                  "disarm"                        -
  isolation                "none"                          -
  layer2                   "1"                             "1"
  online                   "1"                             "1"
  performance_stats        "1"                             -
  portname                 ""                              -
  portno                   "0"                             -
  priority_queueing        "always queue 2"                -
  vnicc/bridge_invisible   "n/a"                           -
  vnicc/flooding           "1"                             "1"
  vnicc/learning           "1"                             "1"
  vnicc/learning_timeout   "600"                           -
  vnicc/mcast_flooding     "0"                             -
  vnicc/rx_bcast           "1"                             -
  vnicc/takeover_learning  "0"                             -
  vnicc/takeover_setvmac   "0"                             -
```

Agenda

- Setup overview
- Virtual network setups
  - MacVTap
  - Linux bridge
  - Open vSwitch
- Network traffic forwarding
  - NAT-based networking
  - Promiscuous mode
  - MAC address registration
  - VNIC characteristics
- **Network setup - Pros and cons**
  - Network workload description
  - MacVTap considerations
  - Open vSwitch considerations
  - Linux Bridge considerations
- Network performance tuning
  - Sysctl settings
  - Offloads, TX queue
  - Receive Packet Steering (RPS)

Network workload description

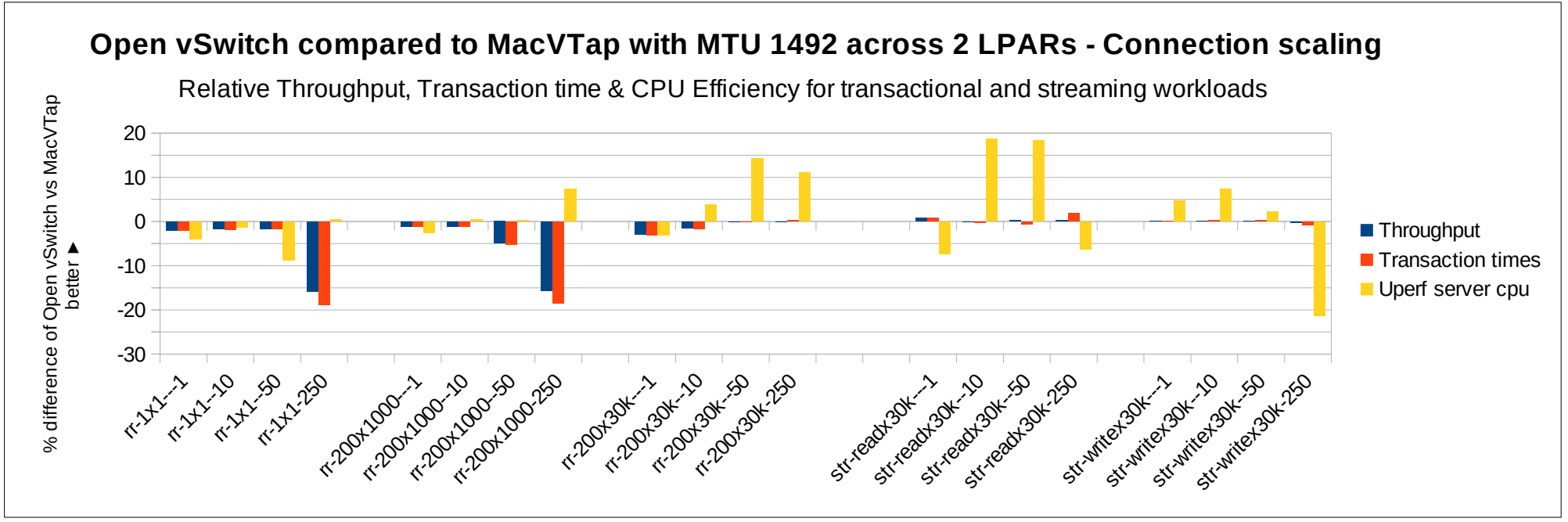
- **Workload**
  - Workload generation was done using uperf (www.uperf.org)

- **Two workload load categories**
  - Transactional workload (RR)
    - 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.

    - Naming convention: {category}-{requestsize}x{responsesize}--{users}
    - Example: rr-200x1000--10 describes a Request-and-Response test
      - sending a 200 byte request
      - receiving a 1000 byte response
      - being generated each by 10 concurrent users (connections)
  - Streaming workloads (STR)
    - 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.
    - 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.

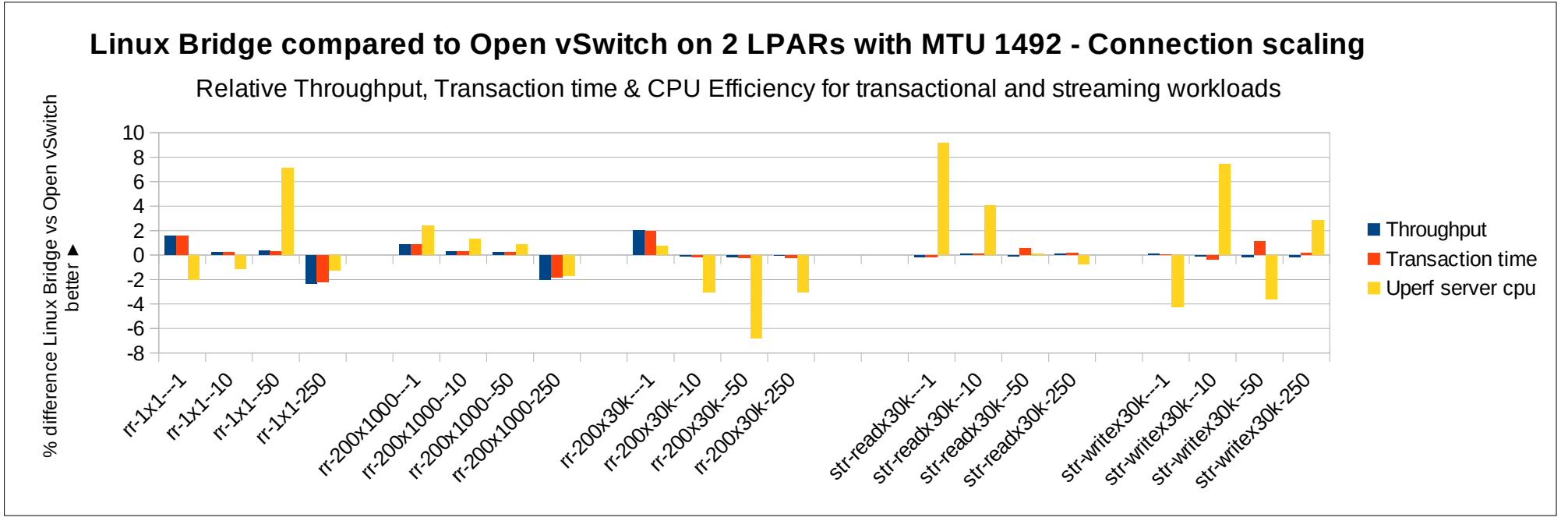
    - Naming convention: {category}-{read|write}x{payloadsize}--{users}
    - Example: str-readx30k--50
      - describes a streaming test read of 30KB datagrams
      - being generated each by 50 concurrent users (connections)

Open vSwitch vs MacVTap



- **Transactional performance observations:**
  - For most workload tests the throughput and latency of Open vSwitch is similar to MacVTap.
  - At 250 users, the latency differences 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
  - Open vSwitch may offer some CPU consumption savings compared to MacVTap for the uperf server, especially for tests with larger payload sizes.

Linux Bridge vs Open vSwitch



- **Transactional and streaming performance observations:**
  - Behavior is nearly identical
- **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.

Network attachment type considerations

- **The following list provides some aspects and does not demand to be complete!**

- **MacVTap**
  - Pro
    - Easy to setup
    - Shortest code path
    - Consistently demonstrated higher throughput and better CPU efficiency
    - Allows sharing of the OSA card between LPARs
    - MAC address registration is handled automatically (changes are propagated)

  - Con
    - No network communication between the KVM host and any of the KVM guests per default
    - It must attach to a physical host interface.
      - It doesn't provide pure virtual networks for KVM guests.
      - Guests exposed directly to the external network
      - Some administrative overhead, changes (e.g. device addresses) must be applied on all MacVTap interfaces
    - No additional features, e.g. no bandwidth management, etc. for the guests

Network attachment type considerations, cont.

- **Open vSwitch with OSA card in promiscuous mode**
  - Pro
    - Performs typically as good or better than a standard Linux bridge.
    - Provides a single point of administration for the network hardware (easy to manage)
    - Very sophisticated and complex network component
      - Supports many more features than does a linux bridge, e.g. channel bonding, QoS management.
      - for details see http://www.openvswitch.org/featureshttp://www.openvswitch.org/features
    - Provides a KVM host isolation mode
      - Does not require a KVM host interface, providing a pure virtual and isolated network.
  - Con
    - Only one LPAR can set the OSA card into the promiscuous mode, the card can not be shared
    - This issue is resolved with the VNIC characteristics on IBM z15

- **Open vSwitch with manual MAC address registration at the OSA card**
  - Pro
    - Same items as the promiscuous mode
    - Allows sharing of the OSA card
    - Applies also to OSA cards in promiscuous mode
  - Con
    - Requires very careful manual management of the registration process, every change must be considered
    - Not supported with VLANs

Network attachment type considerations

- **Linux bridge with NAT**
  - Pro
    - Easy to setup
    - Provides a single point of administration for the network hardware (easy to manage)
    - MAC addresses of the guests do not need to be known to the OSA card
  - Con
    - With NAT is the guest not reachable from the outside
    - No additional features, e.g. no bandwidth management, etc. for the guests

Recommendation

- **For production systems**
  - Open vSwitch with OSA card in promiscuous mode (or VNIC characteristics), especially for large environments
    - a good choice when the restrictions of MacVTap are undesirable.
  - MacVTap
    - if OSA cards need to be shared
    - if short latency is required even under high load

- **For HiperSockets connections**
  - IP routing
  - Linux bridge in NAT mode, if isolation is intended
    The the guests need an additional interface to the outside

- **For smaller test systems, when sharing of OSA cards is a requirement**
  - MacVTap
  - Open vSwitch with manual MAC registration at the OSA card,
    - when Open vSwitch features are needed and VNIC characteristics are not available

Pitfalls

- **MTU size**
  - When using MacVTap or a bridge, the host is not an active participant of the communication path MTU size determination
  - If the MTU size in the host is smaller than the MTU size of the KVM guest
    - that works well as long as the packages sent are smaller than the host MTU size
    - if the KVM guest attempts to send packets that are larger than the MTU size in the host, those packets will be truncated in the host. The host will not fragment the larger packets down!
  - The connection might hang until it times out when the packages are larger than the host MTU size
    - On the sender site: There are no error messages indicating the problem
    - On the receiver site: I/O errors might appear
    - tcpdump reports packages with invalid checksums
  - The host MTU size must be equal to or greater than the MTU size set in the KVM guests!

Agenda

- **Setup overview**
- **Virtual network setups**
  - MacVTap
  - Linux bridge
  - Open vSwitch
- **Network traffic forwarding**
  - NAT-based networking
  - Promiscuous mode
  - MAC address registration
  - VNIC characteristics
- **Network setup - Pros and cons**
  - Network workload description
  - MacVTap considerations
  - Open vSwitch considerations
  - Linux Bridge considerations
- **Network performance tuning**
  - Sysctl settings
  - Offloads, TX queue
  - Receive Packet Steering (RPS)

Sysctl settings

- **Sysctl Settings**

Sysctl Variable	Sysctl Value	Sysctl Value (default)	Comment
net.core.netdev_max_backlog	25000	1000	increase device receive queue
net.core.rmem_max	4136960	262144	Increase TCP/UDP read/write buffers
net.core.wmem_max	4136960	262144	
net.ipv4.tcp_congestion_control	cubic	(often) reno	congestion-avoidance algorithms
net.ipv4.tcp_fin_timeout	1	60	reclaim dead or stale resources
net.ipv4.tcp_low_latency	0	0	to optimize for latency set to 1
net.ipv4.tcp_max_tw_buckets	450000	262144	max number of sockets in “time-wait”
net.ipv4.tcp_tw_reuse	1	0	reuse sockets in the “time-wait” state for new connections.

- **Display sysctl variables**
  - `sysctl -a | less` or `sysctl -a | grep “variable name”` or `sysctl variable1 [variable2] [...]`
- **Setting sysctl values in flight**
  - `echo "value" > /proc/sys/location/variable`
  - `sysctl -w variable=value`
- **Setting sysctl persistently**
  - add/change the value of the variable in /etc/sysctl.conf file: variable=value
  - systemd-sysctl.service needs to be enabled

Increase TX queue, buffers

- **Transmit (TX) queue length**
  - There are two queues
    - The netdev_backlog for receive queue size (sysctl settings)
    - txqueuelen for transmit queue size.
  - Consider to increase the size of the transmit queue
    - The default of 1000 is appropriate for 1 GbE cards
    - For 10 GbE or faster network devices a value of **2500** is good starting point
  - Query with ifconfig or ip
    - `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`
  - Set with
    - ip link set txqueuelen <new-value> dev <interface-name>

- **Increase buffers on the OSA card**
  - Set buffer_count to 128 (default 64)
  - Set in interface udev rule: `ATTR{[ccwgroup/<device addresse>]buffer_count}="128"`
    - For details to udev rules see slide 41

Offloads - Host only

- **Network checksumming - hardware supports checksumming to reduce the CPU load**
  - Set with `ethtool -K <interface> rx on tx on` (uppercase K)
- **TCP segmentation offload (TSO) - hardware supports splitting large packages (send) to MTU size**
  - Scatter gather (SG) - allows buffers from one network package can be distributed over the memory areas
    - SG and tx checksumming are required for TSO
  - Set with `ethtool -K <interface> sg on tx on tso on` (uppercase K)
  - Generic segmentation offload (GSO) - software fallback if usage of TSO is not possible (enabled per default)
    - Set with `ethtool -K <interface> sg on gso on` (uppercase K)
- **Generic receive offload (GRO) - aggregate incomming packages (receive) in software (enabled per default)**
  - Set with `ethtool -K <interface> gro on` (uppercase K)
- **Recommendation is to enable all these features on the KVM host**
  - Set with `ethtool -K <interface> rx on tx on sg on tso on gso on gro on` (uppercase K)
- **Check device settings with**
  - `ethtool -k <interface>` (lowercase k)

Receive Packet Steering (RPS) - Host only

- **Receive Packet Steering (RPS)**
  - Distribute large/high network loads across multiple processor cores.
  - 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.
  - Configured separately for each network device in sysfs.

    `/sys/class/net/<device/queues/rx-<queue#>/rps_cpus`
    - <device> specifies the actual name of the interface device
    - rx-<queue#> represents the network queue number being set
  - Example

    ```
    # cat /sys/class/net/eth0/queues/rx-0/rps_cpus
    0000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000
    ```
    - A comma-delimited bitmap of CPUs.
    - Each number in the bitmap is a hex value and specifies four CPU bit locations.
    - 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.
  - To enable RPS for 1 or more CPUs, the individual bitmask for the selected CPUs must be set to 1.
  - Example for enabling CPU0 - CPU3
    - echo f > /sys/class/net/eth0/queues/rx-0/rps_cpus
    - cat /sys/class/net/eth0/queues/rx-0/rps_cpus
      00,00000000,00000000,0000000f
  - This setting is recommended for the KVM host only.

udev rules (/etc/udev/rules.d) - samples

- **For network are two types of udev rules**
  - the interface: for example 41-qeth-0.0.e000.rules
  - the network: for example 70-persistent-o6s_10g_0.rules
- **Interface udev rule**

```
ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="qeth", GOTO="group_qeth_0.0.e000"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.e000", DRIVER=="qeth", GOTO="group_qeth_0.0.e000"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.e001", DRIVER=="qeth", GOTO="group_qeth_0.0.e000"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.e002", DRIVER=="qeth", GOTO="group_qeth_0.0.e000"
ACTION=="add", SUBSYSTEM=="ccwgroup", KERNEL=="0.0.e000", DRIVER=="qeth", GOTO="cfg_qeth_0.0.e000"
GOTO="end_qeth_0.0.e000"
...
LABEL="cfg_qeth_0.0.e000"
ATTR{[ccwgroup/0.0.e000]layer2}="1"
ATTR{[ccwgroup/0.0.e000]buffer_count}="128"                          << increase buffer count
ATTR{[ccwgroup/0.0.e000]vnicc/flooding}="1"                          << VNIC characteristics
ATTR{[ccwgroup/0.0.e000]vnicc/learning}="1"
ATTR{[ccwgroup/0.0.e000]online}="1"
ATTR{[ccwgroup/0.0.e000]performance_stats}="1"

LABEL="end_qeth_0.0.e000"
```

- **Network rule**

```
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="qeth", KERNELS=="0.0.e000", ATTR{type}=="1" \
,NAME="o6s_10g_0" \
,ATTR{[ccwgroup/0.0.e000]net/o6s_10g_0/queues/rx-0/rps_cpus}="ffffffff" \          << RPS
,RUN+="/sbin/ip link set txqlen 2500 dev '%k'" \                                   << txqlen
,RUN+="/usr/sbin/ethtool -K '%k' rx on tx on sg on tso on gso on gro on"           << offloads
```

  - the '\' must be the last character of the line
  - %k - The kernel name for this device

udev rules - pitfalls

- **On some distributions**
  - udev rules are also placed in the initrd and might overrule the rules in the file system
  - if udev changes do not apply it might be required to rebuild the initrd (or issue a bugzilla)
    - check with lsinitramfs, for example `lsinitramfs /boot/initrd.img | grep 41`

Questions?

- **Further information is located at**
  - KVM Network Performance - Best Practices and Tuning Recommendations
    https://www.ibm.com/support/knowledgecenter/linuxonibm/liaag/wkvm/l0wkvm00_2016.htm
  - Network Storage Protocols in a KVM Environment - NFS/SMB/iSCSI Report
    https://www.ibm.com/support/knowledgecenter/linuxonibm/liaag/wnsp/l0wnsp00_2017.htm
  - Exploiting HiperSockets in a KVM Environment Using IP Routing with Linux on Z - Results and Findings
    https://www.ibm.com/support/knowledgecenter/linuxonibm/liaag/wehs/l0wehs00_2018.htm

Dr. Juergen Doelle

Linux on System z
System Software
Performance Analyst

IBM Deutschland Research & Development
Schoenaicher Strasse 220
71032 Boeblingen, Germany