Linux on IBM z Systems (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.


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.
Agenda

- General Setup
- Virtual Network Setups
  - MacVTap
  - Linux Bridge
  - Open vSwitch
- Network Traffic Forwarding
  - NAT-based networking
  - Promiscuous mode
  - MAC address registration
- Network Setup - Pros and Cons
  - Network Workload Description
  - MacVTap Considerations
  - Open vSwitch Considerations
  - Linux Bridge Considerations
- Network Performance Tuning
  - Sysctl Settings
  - Checksumming, TX Queue
  - Receive Packet Steering (RPS)
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

- General Setup
  - Virtual Network Setups
    - MacVTap
    - Linux Bridge
    - Open vSwitch
  - Network Traffic Forwarding
    - NAT-based networking
    - Promiscuous mode
    - MAC address registration
  - Network Setup - Pros and Cons
    - Network Workload Description
    - MacVTap Considerations
    - Open vSwitch Considerations
    - Linux Bridge Considerations
- Network Performance Tuning
  - Sysctl Settings
  - Checksumming, TX Queue
  - Receive Packet Steering (RPS)
Network Configurations

- There are three virtual network device options for KVM guests
  - MacVTap
  - Linux Bridge
  - Open vSwitch

- Three 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

  - Note:
    - One (and only one) from the options I – III must be selected for a certain guest network interface
    - A guest can have multiple interfaces with different connection types

- Resulting network configurations to OSA cards

<table>
<thead>
<tr>
<th></th>
<th>MacVTap</th>
<th>Linux Bridge</th>
<th>Open vSwitch</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSA promiscuous mode</td>
<td>--</td>
<td>one of these</td>
<td>one of these</td>
</tr>
<tr>
<td>OSA Interface registration</td>
<td>implicitly</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAT</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

- Another option is to connect the guest to the host network (no assigned physical interface)
  - IP routing → handled in a separate presentation
Important Hints to Network Configurations

- **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

- **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.
Agenda

- General Setup
  - Virtual Network Setups
    - MacVTap
    - Linux Bridge
    - Open vSwitch
  - Network Traffic Forwarding
    - NAT-based networking
    - Promiscuous mode
    - MAC address registration
  - Network Setup - Pros and Cons
    - Network Workload Description
    - MacVTap Considerations
    - Open vSwitch Considerations
    - Linux Bridge Considerations
  - Network Performance Tuning
    - Sysctl Settings
    - Checksumming, 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
  1. **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 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

- To load the driver
  - `modprobe macvlan`

- Domain XML statements

  ```xml
  <interface type="direct" dev="vnic0">
    <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" dev="vnic0">`
    - A direct attachment to an existing KVM host network device.
    - The interface will show up with name vnic0 in the guest
    - 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
    - And the 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).

- 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
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" dev="vnic0">
    <source bridge="bridge-name"/>
    <target dev="my-vnet0"/>
    <model type="virtio"/>
    <driver name="vhost"/>
</interface>
```

This will create in the guest

- A new network interface with name vnic0
- Connected in the host via the interface my-vnet0
- Against bridge bridge-name

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> <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>`
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

```xml
<interface type="bridge" dev="vnic0">
  <source bridge="bridge-name"/>
  <virtualport type="openvswitch"/>
  <target dev="my-vnet0"/>
  <model type="virtio"/>
  <driver name="vhost"/>
</interface>
```

- This will create in the guest
  - A new network interface with name vnic0
  - Connected in the host via the interface my-vnet0
  - Against bridge bridge-name

- 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"
```
OpenvSwitch – More Information

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

Especially
Agenda

- General Setup
- Virtual Network Setups
  - MacVTap
  - Linux Bridge
  - Open vSwitch
- Network Traffic Forwarding
  - NAT-based networking
  - Promiscuous mode
  - MAC address registration
- Network Setup - Pros and Cons
  - Network Workload Description
  - MacVTap Considerations
  - Open vSwitch Considerations
  - Linux Bridge Considerations
- Network Performance Tuning
  - Sysctl Settings
  - Checksumming, 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**
  - Registers and manages implicitly the MAC address at the OSA card (no actions required)

- **OpenvSwitch/Linux bridge**
  - By default, the OSA card only forwards network traffic destined to devices that are known
  - There are two 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
  - 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
    ```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.2" end="192.168.X.254"/>
        </dhcp>
      </ip>
    </network>
    ```
  - Change `<name>`, `<bridge name>` and `<ip address>` to suite your needs.
  - 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
- 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
Agenda

- General Setup
- Virtual Network Setups
  - MacVTap
  - Linux Bridge
  - Open vSwitch
- Network Traffic Forwarding
  - NAT-based networking
  - Promiscuous mode
  - MAC address registration
- Network Setup - Pros and Cons
  - Network Workload Description
  - MacVTap Considerations
  - Open vSwitch Considerations
  - Linux Bridge Considerations
- Network Performance Tuning
  - Sysctl Settings
  - Checksumming, TX Queue
  - Receive Packet Steering (RPS)
Network Workload Description

- **Workload**
  - Workload generation was done using uperf ([www.uperf.org](http://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}{1c}-{requestsizer}{responsesizer}--{users}`
      - Example: `rr1c-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)
MacVTap Considerations

- **MacVTap driver consistently demonstrated higher throughput and better CPU efficiency**

- **Some limitations**
  - No network communication between the KVM host and any of the KVM guests per default
    - Might be overcome with a second MacVTap interface to the host with a different network subnet
    - MacVTap only restricts traffic flow to the same subnet shared between host and guest.
  - It must attach to a physical host interface. KVM guests using MacVTap will be exposed to external network traffic.
    - It doesn't provide pure virtual networks for KVM guests that bridges allow.
  - Some administrative overhead, changes must be applied on all MacVTap interfaces
Open vSwitch Considerations

- **Open vSwitch is a good choice when the restrictions of MacVTap are undesirable.**
  - Performs typically as good or better than a standard Linux bridge.
  - Provides a single point of administration

- **Very sophisticated and complex network component**
  - Supports many more features than does a Linux bridge, e.g. channel bonding, QoS management.
  - [http://www.openvswitch.org/features/](http://www.openvswitch.org/features/)

- **Provides a KVM host isolation mode**
  - Does not require a KVM host interface, providing a pure virtual and isolated network.
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, 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.
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
    - Allows sharing of the OSA card between LPARs
    - MAC address registration is handled automatically (changes are propagated)
  - **Con**
    - Changes on the network hardware (e.g. device addresses) need to be applied to all connected guests
    - No additional features, e.g. no bandwidth management for the guests
    - Guests exposed directly to the external network

- **Linux bridge with NAT**
  - **Pro**
    - Easy to setup
    - Single point to connect to the network hardware (easy to manage)
    - MAC addresses of the guests do not need to be known to the OSA card
  - **Con**
    - No bandwidth management for the guests
    - With NAT is the guest not reachable from the outside
Network Attachment Type Considerations, cont.

- **Open vSwitch with OSA card in promiscuous mode**
  - **Pro**
    - Single point to connect to the network hardware (easy to manage)
    - Very powerful component with many additional features, e.g. bonding support, workload management
      for details see [http://www.openvswitch.org/features](http://www.openvswitch.org/features)
  - **Con**
    - Only one LPAR can set the OSA card into the promiscuous mode, the card can not be shared

- **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
Recommendation

- For production systems
  I. Open vSwitch with OSA card in promiscuous mode, especially for large environments
  II. MacVTap, if OSA cards need to be shared

- For HiperSockets connections
  I. IP routing
  II. 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
  I. MacVTap
  II. Open vSwitch with manual MAC registration at the OSA card, when Open vSwitch features are needed
Pitfalls

- **MTU size**
  - When using MacVTap or a bridge with a direct connection to a host interface, the host is not an active participant of the communication path MTU size determination.
  - That means:
    - 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.
  - That works well as long as the packages are smaller than the host MTU size.
  - The connection will hang until it times out when the packages are larger than the host MTU size:
    - there are no error messages indicating the problem,
    - the only indicator are packages with invalid checksums reported from tcpdump.

- The host MTU size must be equal to or greater than the MTU size set in the KVM guests.
Agenda

- General Setup
- Virtual Network Setups
  - MacVTap
  - Linux Bridge
  - Open vSwitch
- Network Traffic Forwarding
  - NAT-based networking
  - Promiscuous mode
  - MAC address registration
- Network Setup - Pros and Cons
  - Network Workload Description
  - MacVTap Considerations
  - Open vSwitch Considerations
  - Linux Bridge Considerations
- Network Performance Tuning
  - Sysctl Settings
  - Checksumming, TX Queue
  - Receive Packet Steering (RPS)
## Sysctl Settings

### Display sysctl variables
- `sysctl -a | less` or `sysctl -a | grep “variable name”` or `sysctl variable1 [variable2] [variable3]` ...

### 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`

---

<table>
<thead>
<tr>
<th>Sysctl Variable</th>
<th>Sysctl Value</th>
<th>Sysctl Value (default)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>net.core.netdev_max_backlog</td>
<td>25000</td>
<td>1000</td>
<td>increase device receive queue</td>
</tr>
<tr>
<td>net.core.rmem_max</td>
<td>4136960</td>
<td>262144</td>
<td>Increase TCP read/write buffers</td>
</tr>
<tr>
<td>net.core.wmem_max</td>
<td>4136960</td>
<td>262144</td>
<td></td>
</tr>
<tr>
<td>net.ipv4.tcp_congestion_control</td>
<td>cubic</td>
<td>(often) reno</td>
<td>congestion-avoidance algorithms</td>
</tr>
<tr>
<td>net.ipv4.tcp_fin_timeout</td>
<td>1</td>
<td>60</td>
<td>reclaim dead or stale resources</td>
</tr>
<tr>
<td>net.ipv4.tcp_limit_output_bytes</td>
<td>131072</td>
<td>262144</td>
<td>improve latency</td>
</tr>
<tr>
<td>net.ipv4.tcp_low_latency</td>
<td>0</td>
<td>0</td>
<td>to optimze for latency set to 1</td>
</tr>
<tr>
<td>net.ipv4.tcp_max_tw_buckets</td>
<td>450000</td>
<td>262144</td>
<td>max number of sockets in &quot;time-wait&quot;</td>
</tr>
<tr>
<td>net.ipv4.tcp_rmem</td>
<td>4096 87380</td>
<td>4096 87380 6291456</td>
<td>min, default, max size of TCP socket receive buffer</td>
</tr>
<tr>
<td>net.ipv4.tcp_twReuse</td>
<td>1</td>
<td>0</td>
<td>reuse sockets in the “time-wait” state for new connections.</td>
</tr>
<tr>
<td>net.ipv4.tcp_wmem</td>
<td>4096 16384</td>
<td>4096 16384 4194304</td>
<td>min, default, max size of TCP socket send buffer</td>
</tr>
</tbody>
</table>
Increase TX Queue

- Transmit (TX) Queue length
  - Consider to increase the size of the send queue between
    - The kernel network subsystems and the driver for network interface card
    - The default of 1000 is appropriate for 1 GiBit cards
    - For 10 GiBit or faster network devices a value of 2500 is good starting point
  - There are two queues
    - The netdev_backlog for receive queue size (sysctl settings)
    - txqueuelen for transmit queue size.
  - 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>
Checksumming, OSA Buffers (Host only)

- **Network checksumming**
  - The OSA network cards support two checksumming options.
    - Software checksumming
    - Checksumming on the hardware (on the OSA card).
  - Set to hardware checksumming to reduce the CPU load and perform checksumming faster
  - Change in “ifcfg-{interface}” file for the target interface.
    - Modify the OPTIONS statement add “checksumming=hw_checksumming”
    - After making this change, the interface must be restarted

- **Increase buffers on the OSA card**
  - Set buffer_count to 128 (default 64)
  - Change in “ifcfg-{interface}” file for the target interface.
    - Modify the OPTIONS statement add “buffer_count=128”
    - After making this change, the interface must be restarted

- **Final suggestion for the OPTIONS statement for OSA cards in the KVM host**
  - OPTIONS="checksumming=hw_checksumming buffer_count=128 layer2=1"
Receive Packet Steering (RPS)

- **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`
        - 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.
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

- Thanks to Mark Peloquin from the Cognitive Systems SW Development!

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

Dr. Juergen Doelle
Linux on System z
System Software Performance Analyst