Automating Virtual Machine Network Profiles

Vivek Kashyap
IBM
kashyapv@us.ibm.com

Arnd Bergman
IBM
arndb@de.ibm.com

Stefan Berger
IBM
stefanb@us.ibm.com

Gerhard Stenzel
IBM
Gerhard.Stenzel@de.ibm.com

Jens Osterkamp
IBM
Jens.Osterkamp@de.ibm.com

Abstract

With the explosion of use of virtual machines in the data-center/cloud environments there is correspondingly a requirement for automating the associated network management and administration. The virtual machines share the limited number of network adapters on the system among them but may run workloads with contending network requirements. Furthermore, these workloads may be run on behalf of customers desiring complete isolation of their network traffic. The enforcement of network traffic isolation through access controls (filters) and VLANs on the host adds additional run-time and administrative overhead. This is further exacerbated when Virtual Machines are migrated to another physical system as the corresponding network profiles must be re-enforced on the target system. The physical switches must also be reprogrammed.

This paper describes the Linux enhancements in kernel, in libvirt and layer-2 networking, enabling the offloading of the switching function to the external physical switches while retaining the control in Linux host. The layer 2 network filters and QoS profiles are automatically migrated with the virtual machine on to the target system and imposed on the physical switch port without administrative intervention.

We discuss the proposed IEEE standard (802.1Qbg) and its implementation on Linux for automated migration of port profiles when a VM is migrated from one system to another.

1 Introduction

A network adapter is shared across multiple virtual machines(VM) on a Linux host. To accommodate VM-VM and VM to external network communication a virtual bridge is included in the Linux host. The Linux virtual bridge relays unicast traffic between the virtual and physical interfaces attached to it. It further replicates and forwards broadcast and multicast transmission received on its port(s).

For network isolation and security, the iptables/ebtables rules might be enforced on the system. The more common rules are to prevent ARP or IP spoofing, block sending a link level broadcast, or to allow only specific set of protocol traffic.

Different workloads running in the VMs might also be specifically restricted within certain limits based on the workload requirements, priorities and possibly based on the bandwidth purchased by the user.

For the purposes of this paper the layer-2 forwarding, multiplexing and filtering(ebtables) function is collectively considered part of the virtual-bridge.

The per-packet processing for bridging function - packet relaying, replicating, evaluation against filter rules, or bandwidth control - imposes a heavy burden that takes away CPU resources that could be utilized more gainfully. This problem gets more and more exacerbated as the number of virtual machines supported on a single host increases in the multi-core, and large memory systems being used in the data-center/cloud environments.

Another problem faced in large deployments is the need for maintaining consistent view of the port profiles. The switch fabric and the embedded switches in the hypervisor(Linux KVM host) may have different capabilities and also be under separate administrative control.

This is further exacerbated as the VMs (running important workloads) migrate from one system to another.

The filter rules and QoS enforcement must follow the VM and be quickly in place. Not all policies may be deployable in the hypervisor and the switch must be informed of the migration.

As the MAC address associated with the virtual interfaces migrates as well, the physical switch at the target does not know if the MAC, now visible at another port, is a migrated VM or a different VM using the same MAC. Thereby, it needs to be informed of the port-profile policy to deploy.

The proposal 'Edge Virtual Bridging' in IEEE [802.1Qbg] addresses these issues through offloading of the switching function to the adjacent bridge i.e. the physical switch in the network.

1.1 Edge Virtual Bridging: IEEE 802.1Qbg

The Edge Virtual Bridging(EVB) proposal defines the protocols, configuration and control required across the physical end station, such as the Linux host, and the adjacent switch.

This section provides an introduction to the IEEE 802.1Qbg proposal and the subsequent sections describe our implementation of the same on Linux to configure KVM guests in 'VEPA' mode.

Reflective Relay

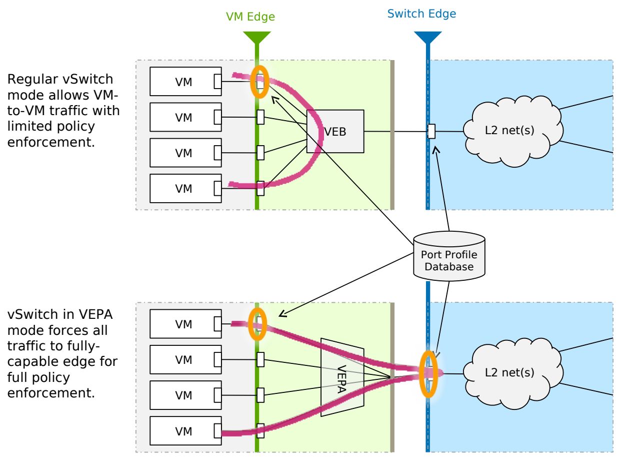
The proposed standard extends the adjacent bridge capabilities to the virtual machines by ensuring that all the packets sent by the VM's are first sent to the switch port. The switch then consults its tables, and if the packet is destined to another VM on the same host, sends the packet back to the host. The packet is then forwarded to the destination VM.

This packet flow enables the switch to enforce fabric wide packet filtering and control policies across all network traffic. Otherwise the policy and rules might need to be co-ordinated and enforced across the switch and the hypervisor leading to administrative overhead and inefficiencies outlined earlier.

Existent switches do not reflect the packet back on the port on which it was received. Therefore, a new mode, referred to as the 'reflective relay' mode is introduced in the Edge Virtual Bridging(EVB) proposal.

Virtual Ethernet Port Aggregator

The component in the physical end-station that works together with the adjacent switch to support EVB is called "Virtual Ethernet Port Aggregator" or VEPA.

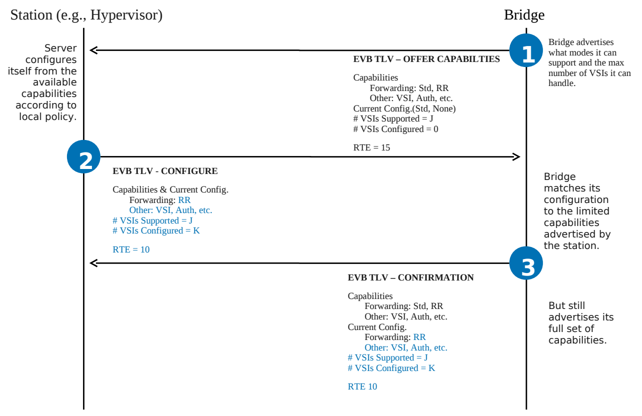

VM Edge
Switch Edge
Regular vSwitch mode allows VM-to-VM traffic with limited policy enforcement.
VM
VM
VM
VM
VEB
L2 net(s)
Port Profile Database
vSwitch in VEPA mode forces all traffic to fully-capable edge for full policy enforcement.
VM
VM
VM
VM
VEPA
L2 net(s)

A VEPA is very simplified version of an Ethernet bridge that allows multiple downlink ports to communicate with a single uplink port but not with each other. Ethernet frames from one of the downlink ports get sent directly to the uplink, and Ethernet frames arriving at the uplink port get forwarded to just the destination with the matching MAC address, or flooded to all downlink ports in case of broadcast. A VEPA does not support unknown unicast frames, which get silently dropped.

The VEPA component therefore, is able to provide VM to VM communication in conjunction with an uplink port which is configured in 'reflective relay' mode.

Dynamic discovery and configuration for VEPA mode

The Link-level Discovery Protocol(LLDP)[LLDP] is used for layer-2 discovery operations. The EVB proposal extends the TLVs (configuration messages) supported to include advertisement of 'reflective relay' capabilities and setting of the adjacent bridge's port in reflective relay mode. The TLV also includes additional parameters such as the the number of virtual interfaces (VSI or Virtual station interfaces in EVB parlance) that a station or bridge can support. See [802.1Qbg] for description of all capabilities and parameters exchanged.

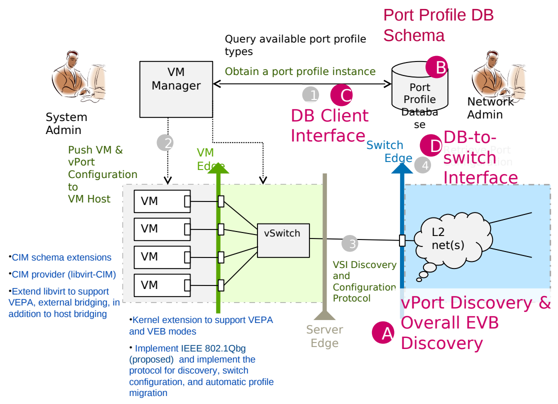

Station (e.g., Hypervisor)
Bridge
1
Bridge advertises what modes it can support and the max number of VSIs it can handle.
EVB TLV – OFFER CAPABILTIES
Capabilities
Forwarding: Std, RR
Other: VSI, Auth, etc.
Current Config.(Std, None)
# VSIs Supported = J
# VSIs Configured = 0
RTE = 15
Server configures itself from the available capabilities according to local policy.
2
EVB TLV - CONFIGURE
Capabilities & Current Config.
Forwarding: RR
Other: VSI, Auth, etc.
# VSIs Supported = J
# VSIs Configured = K
RTE = 10
Bridge matches its configuration to the limited capabilities advertised by the station.
3
EVB TLV – CONFIRMATION
Capabilities
Forwarding: Std, RR
Other: VSI, Auth, etc.
Current Config.
Forwarding: RR
Other: VSI, Auth, etc.
# VSIs Supported = J
# VSIs Configured = K
RTE 10
But still advertises its full set of capabilities.

The bridge periodically advertises its capabilities. The Station receives the TLV, and based on its configuration, may respond to configure the port in 'reflective relay' (RR) mode.

VSI Discovery and Configuration

The switching function is offloaded from the end-station with VEPA and dynamic configuration of the switch port to RR mode. This however does not fully address the problems outlined above. A mechanism is required to associate the virtual interface (VSI) to specific profile for filtering, VLAN and bandwidth control.

This is achieved by informing the switch of the MAC address and VLAN ID pair and the profile that must be used.


Port Profile DB Schema
Query available port profile types
Obtain a port profile instance
VM Manager
B
Port Profile Database
Network Admin
System Admin
1
C
DB Client Interface
2
Push VM & vPort Configuration to VM Host
VM Edge
Switch Edge
D
DB-to-switch Interface
4
VM
VM
VM
VM
vSwitch
3
L2 net(s)
VSI Discovery and Configuration Protocol
vPort Discovery & Overall EVB Discovery
A
Server Edge
•CIM schema extensions
•CIM provider (libvirt-CIM)
•Extend libvirt to support VEPA, external bridging, in addition to host bridging
•Kernel extension to support VEPA and VEB modes
• Implement IEEE 802.1Qbg (proposed) and implement the protocol for discovery, switch configuration, and automatic profile migration

The bridge on receiving the association request gets the profile details from a database and configures its port accordingly. This mechanism also addresses the issue of 'reincarnated' or reused MAC address and a MAC address that has appeared on the port after a VM migration since the switch is informed of the exact profile to impose.

The VSI Discovery protocol (VDP) defined by EVB, therefore defines a set of states and commands exchanged between the bridge and the station. These are:

- *Pre-associate:* Inform the switch of the VSI and port-profile and intention to associate
- *Pre-associate with Resource Reservation:* Same as Pre-associate except also reserve resources at the switch for a future association.
- *Associate:* Associate the VSI. This causes all resources to be allocated at the switch and the imposition of the VSI port profile.
- *DeAssociate:* De-associate the VSI from the port and profile

Edge Control Protocol

The discovery and exchange of bridge capabilities is performed over LLDP. LLDP is an unacknowledged protocol and has limitations on frequency and number of transmissions.

For VDP, the EVB has proposed a new link-level transport called the "Edge Control Protocol"(ECP) which acknowledges the messages exchanged. This enables the End Station to transmit discovery operations more frequently. The VDP protocol messages will be carried in TLVs over ECP. The TLVs carry the VDP state requests and responses.

2 Design and Implementation

The mapping of IEEE protocol to the Linux implementation can be broken into a few distinct interlocked components. These are:

- Extend Linux kernel to support 'VEPA' mode
- Extend libvirt interface xml to define VEPA mode and VSI state
- Implement user-space daemon to support EVB link level protocols

2.1 MACVTAP: Supporting a 'VEPA' interface

When applied to Linux, a VEPA lets us share a single Ethernet NIC between multiple KVM guests that all get access to the Ethernet segment, but without the need to set up an actual bridge that has both administrative and performance overhead associated with features like MAC address learning, spanning tree protocol or filtering.

For our needs, we developed our solution over the pre-existing 'macvlan' driver supported in Linux. Macvlan provides virtual Ethernet interfaces that can be created using the "ip link" command and that can be used by applications, virtual machines or containers just like any other Ethernet interface.

One significant drawback for our needs in macvlan implementation was that it cannot easily be connected to Qemu/KVM, which expects a tun/tap device instead of a network interface. In addition, the VEPA implementation has to ensure that broadcast and multicast frames never get delivered to the source port but do get forwarded to all other ports that want them.

In order to connect macvlan devices to a kvm virtual machine, a "macvtap" device driver was implemented. Macvtap plugs into the macvlan device driver, and lets each of its downstream ports show up in the system as a character device rather than a network interface. This character device implements a subset of the API of the tun/tap driver that is typically used to connect a KVM guest to the host network.

In the implementation of macvtap, a few shortcuts could be taken compared to the combination of tun/tap with a bridge connected to an external NIC. Most importantly, frames sent from the guest to the tap are not injected into the receive path of the host but directly into the transmit queue of the outbound interfaces. Similarly, inbound frames do not need to get received by the host and then sent out to a tun/tap device but simply get put into the guests receive path when they get intercepted by the receive function of the uplink interface.

2.2 libvirt:VEPA interface for the KVM guest

With the necessary infrastructure for VEPA in place with the macvtap/macvlan implementation we needed to integrate the capability into the libvirt domain xml.

Specifying the VEPA interface

Since the macvlan/macvtap are tied specifically to an interface that provides the uplink port for the 'VEPA' function we decided to provide a strong linkage between the macvtap interfaces and the backing up device.

This is therefore specified in the guest's domain definition as described below. The example assumes the use of 'eth0' as the source device.

```
<interface type='direct'>
  <source dev='static' mode='vepa'/>
  <model type='virtio'/>
</interface>
```

Libvirt creates a macvtap interface when a virtual machine with a direct network interface type is started or such an interface type attached to a running virtual machine.

Libvirt opens the tap device to get a filedescriptor for Qemu to write packets to and receive packets from. A macvtap device works similar to a tap device in that an interface is created on the host and a filedescriptor is subsequently passed to Qemu. However, a macvtap device requires more active management by libvirt during device teardown. Whereas for a tap device it is sufficient that Qemu closes the tap filedescriptor for the tap device's network interface in the host to disappear, libvirt must actively tear down the macvtap device after Qemu was detected to have terminated. Creation and teardown of a macvtap device is done using netlink messages that libvirt sends to the kernel device driver. The command line parameters for Qemu are the same as for a tap device and pass the filedescriptor.

```
[...]  -net nic,
macaddr=52:54:00:0c:dd:47,
vlan=1,model=virtio,
name=net1 -net tap,fd=15,
vlan=0,name=hostnet1
[...]
```

Specifying the VSI state information

The VSI state comprising of the profile to be imposed at the switch port is specified in the interface description as well. The VSI state along with the MAC address, and the VLAN id is therefore forwarded to the user-space daemon implementing the VDP/ECP protocols.

As of writing of this paper we are working to extend the LLDPAD[e1000] daemon to support both the EVB TLVs for RR mode, the ECP and VDP protocols.

The libvirt daemon is extended to send netlink messages with the relevant details to the protocol daemon, LLDPAD. The daemon will then initiate the setup protocol with the switch to enable the packet flow. For this LLDPAD will register the MAC/VLAN pair with the switch along with the VSI data. The success (or failure) will be reported back to libvirt and the guest will be brought online (or failed). In case of failure, libvirt will not start the VM or declare the hot-plugging of an interface to have failed.

The VSI state is specified as in the following example:

```
<interface type='direct'>
  <source dev='static' mode='vepa'/>
  <model type='virtio'/>
  <vsi managerid='12'
   typeid='0x123456'
   typeidversion='1'
   instanceid='insert-uuid-here' />
</interface>
```

VSI states

The ongoing libvirt extensions will send the 'Associate' command when the guest is being brought online and will send a 'DeAssociate' command when the guest is shutdown or suspended or has been migrated.

3 Emulating VEPA bridge

As of this writing there are no switches in the market that support the 802.1Qbg VEPA or reflective relay mode. Therefore, for testing purposes, we utilized the support for 'hairpin mode' or reflective-relay mode already included in the Linux kernel. This enables us to test the raw packet flow that will occur after the switch setup protocol has been successfully completed. Testing of the switch setup protocol will need to be done later.

```
brctl addif <bridge> <interface>
cd /sys/class/net/<interface>/brport
echo 1 > hairpin_mode
```

The VEPA enabled system was then be tested against the Linux host configured as above.

4 Future Work

The libvirt and the LLDPAD work described above are still in progress. The implementation will cover the EVB TLVs, ECP and VDP protocols. The 802.1Qbg proposal also describes 'multi-channel' support and the corresponding 'Channel Discovery and Configuration Protocol' (CDCP). We will extend our implementation to support CDCP in the future.

A large set of management applications use the DMTF's CIM protocol to discover, create and manage virtual machines. There is work ongoing to implement libvirt-CIM providers such that network profile automation can be managed by CIM clients.

Migration of VMs

With the current implementation, at the time of the migration the target system must be put into 'Associate' state with the switch when the migration commences since there is no hook or mechanism to insert additional function in the migration process as implemented by qemu/kvm. It would be advantageous to be able to insert a 'PreAssociate' state on the target and then move to 'Associate' state only when the source is suspended. This will avoid overlapping associates from two systems at the same time since that can cause unpredictable results in the layer-2 fabric if the source and target are connected to the same switch. This issue is mitigated since migrating VMs are only active in one place, either on the source or the target host, but never on both hosts at the same time.

5 Acknowledgements

We would like to thank the EVB working group, IEEE authors group and the Linux community members with whom we have worked closely to define and implement the 802.1Qbg standard. We would specifically like to thank Renato Recio, Mike Krause, Rakesh Sharma Jeff Lynch, Daniel Berrange, Daniel Veillard, and Chris Wright for various discussions on protocol development, implementation and code reviews. Thanks to Anna Fischer for enabling the "hairpin mode" in the Linux bridge.

6 References

[802.1Qbg] http://www.ieee802.org/1/pages/802.1bg.html

[LLDP] http://standards.ieee.org/getieee802/download/802.1AB-2005.pdf

[e1000] http://e1000.sf.net

Proceedings of the Linux Symposium

July 13th–16th, 2010
Ottawa, Ontario
Canada