Virtual Ethernet Switches

Ross Alexander

February 11, 2018

Contents

1 Linux Bridge and KVM

We start with Linux, partly because ESX used to run on it, and partly because it exposes may of the concepts to the user rather than behind the scenes.

The most common hypervisor for Linux is KVM, which uses the QEMU emulator with an in kernel module. QEMU emulates a number of PCI NICs include Intel e1000 and virtio-net. Ethernet frames generated by the guest kernel are passed to the virtual NIC. These need to be forwarded to other guests and the outside world.

Linux (and other Unix flavours) have the concept of a TAP device. This acts like a standard Ethernet NIC but instead of being attached to a hardware device is attached to a running process, in this case the kvm process running the guest.

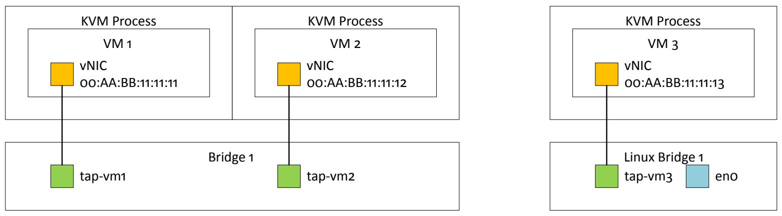

Figure 1: Linux VMs with KVM

Within Linux there is a bridge device, which works as a standard L2 (MAC) learning bridge. The bridge module allows for the creation of multiple, independent, bridges, to enable traffic isolated. In the example two KVM guests can communicate with each other but they cannot communicate with with the third guest, which resides on a second hypervisor.

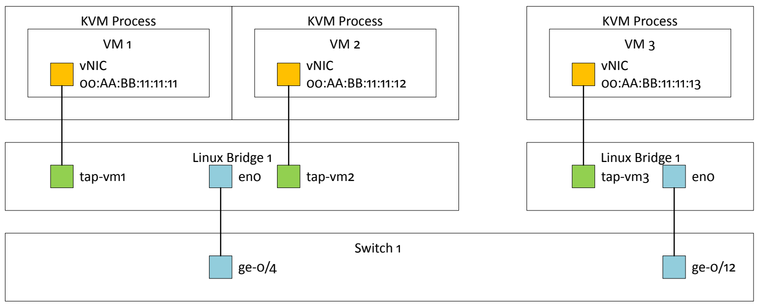

Figure 2: KVM connected to phyiscal switch

We can attach a physical NIC to the virtual bridges while cabling it to a physical switch to enable cross hypervisor connectivity. Logically this is three switch network with three hosts attached. The switches all support L2 Learning and may be running spanning tree.

1.1 L2 (MAC) Learning rules

L2 (or MAC) learning is fundamental to the forwarding of Ethernet frames within an L2 network (or domain). It is possible to not use L2 learning provided a frame is never forwarded back out of the ingres port but would be extremely inefficient.

1. Record ingres port of frame.
2. Update L2 FIB with source MAC and ingres port.
3. If unicast search destination port in L2 FIB using destination MAC address, otherwise egress is flood.
4. If FIB entry found record egress port, otherwise mark to flood.
5. If flood then replicate frame to all egress ports except the ingres port.
6. If egress == ingres then drop frame rather than transmit.

1.2 VLANs for L2 seperation

Traditional L2 networking uses VLANs for traffic separation. The current standard is 802.1q, which uses a 12 bit VLAN ID (VLID). A common configuration for switches is for ports to either be access (untagged) ports or trunk (tagged) ports. Access ports can only belong to a single VLAN while trunked ports belong to a number of VLANs. The treatment of untagged frame on a trunk port and tagged frame on an access port is vendor specific.

Specially the Linux bridge removes an 802.1q tags (or the outer tag for Q-in-Q) on ingres and never adds an tag on egress. Therefore it is necessary to have a bridge per VLAN used. It is possible to create slave devices so any ethernet device can effectively become an access port while its parent becomes a trunk device. For example if our NIC is en0 then we can create a device en0.1200, which will add a VLAN tag on ingres traffic and sent it out of en0 while traffic arriving for VLAN 1200 on en0 will be egressed out of en0.1200 with the tagged removed.

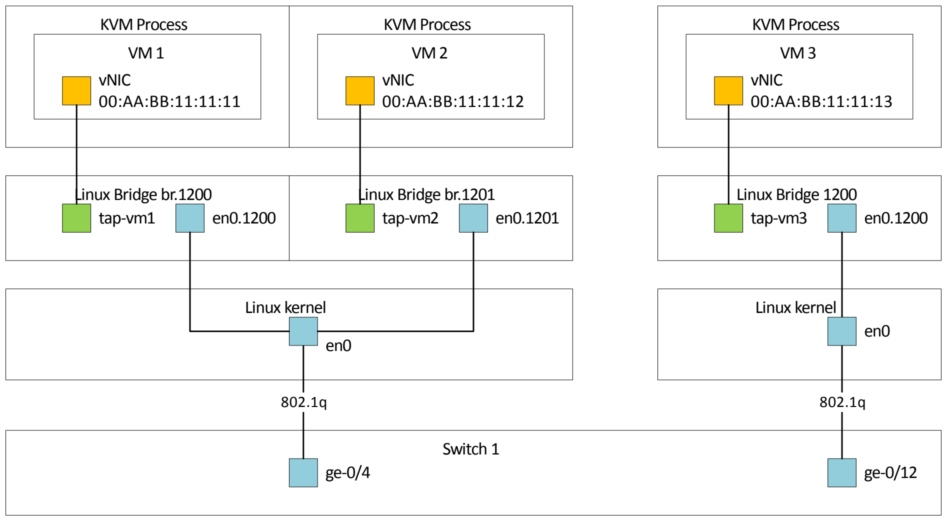

Figure 3: KVM with VLANs

2 vmware vSwitch

2.1 Standard vSwitch

The vmware ESXi hypervisor implements two virtual switches. The vNetwork Standard vSwitch (vss) and the vNetwork Distributed vSwitch (vds). All ESXi hosts support the standard vSwitch (in fact this needs to be configured for even management traffic) while dvs requires a vCenter implementation (and additional licensing).

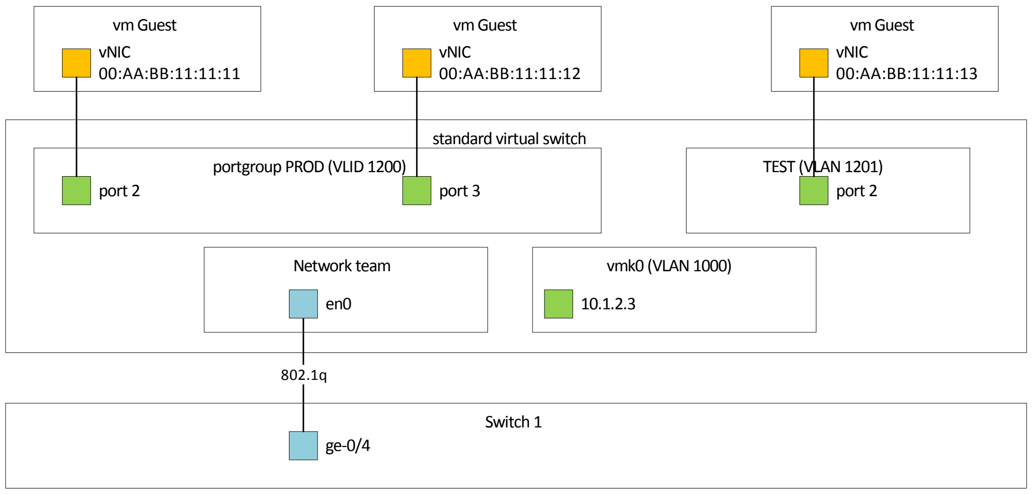

Figure 4: Standard vSwitch

A standard vswitch consists of one or more portgroups and one or more uplink devices. It is possible to attach an L3 (IPv4 or IPv6) interface to a vswitch as well, known as a vmk (vm kernel) interface. Standard vswitches support 802.1q VLANs for separation.

Standard vswitches need to be configured on each ESXi host separately. Within a vCenter environment they can be managed centrally but every ESXi needs to have atleast one vmk interface and this must be configured initially on the console. Once a hypervisor is in vCenter it is possible to attach the vmk to a distributed vswitch.

2.2 Distributed vSwitch

In a large environment (ie beyond a couple of hypervisors) maintaining standard vswitches for production networks becomes problematic. The solution is the distributed vswitch. This is indentical to a standard vswitch for configuration except the only the number of uplink ports are specified in advance.

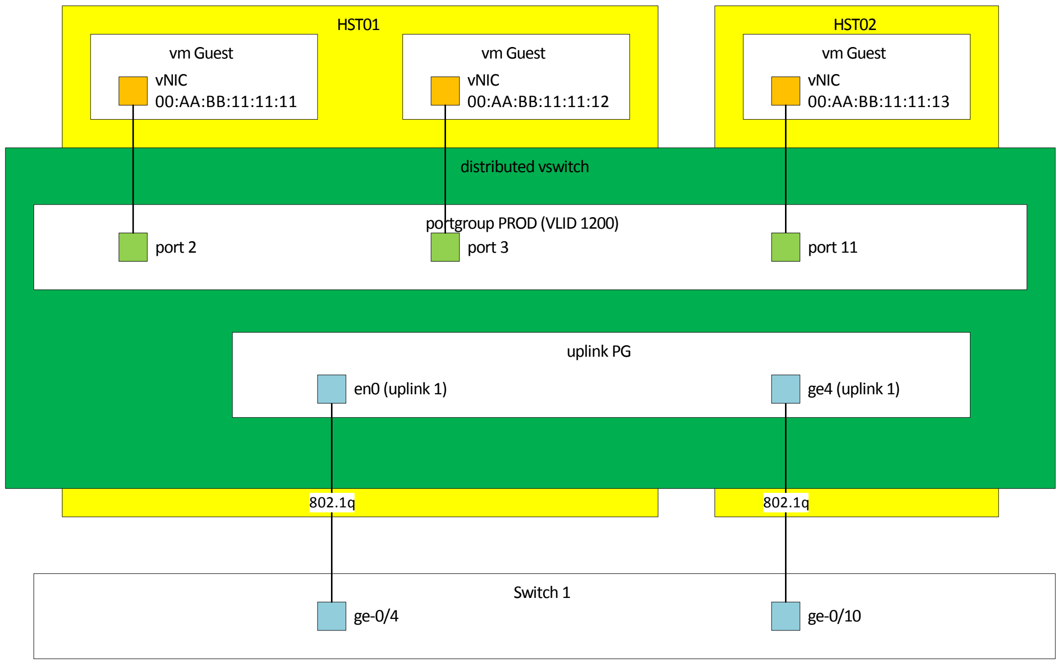

Figure 5: Distributed vSwitch

When a dvs is attached to a host the some (or all) of the host's NICs are mapped to the uplinks on the dvs. This allows hosts to have different physical NICs (or example one host may have four 1gb ports while another may have two 10gb ports). Once a dvs has been attached to host all the portgroups in the vSwitch are accessible by guests running on that host.

The distributed vswitch also has a number of extra features over the standard vswitch, such as Netflow, port mirroring and LLDP.

3 vSwitches with multiple uplinks

For resilience (and performance) most vSwitches are generally have more than one uplink. If these uplinks where treated as separate interfaces then without Spanning Tree Protocol (STP) it would be possible to create an L2 loop. With the Linux bridge this can be done, both with and without STP. However caution is required.

To avoid the possibility of the vSwitch creating a loop all the uplinks are treated as a single teamed (or bonded) interface. L2 forward rules always block traffic egressing from the same ingres port, so a vswitch will never forward a frame from the physical network back out again.

Linux has a similar concept, known as a bond device, and this is used by OVM. Both the linux bond and vswitch team have a number of modes of operation. These can be classified into two groups, switch dependant, and switch independant.

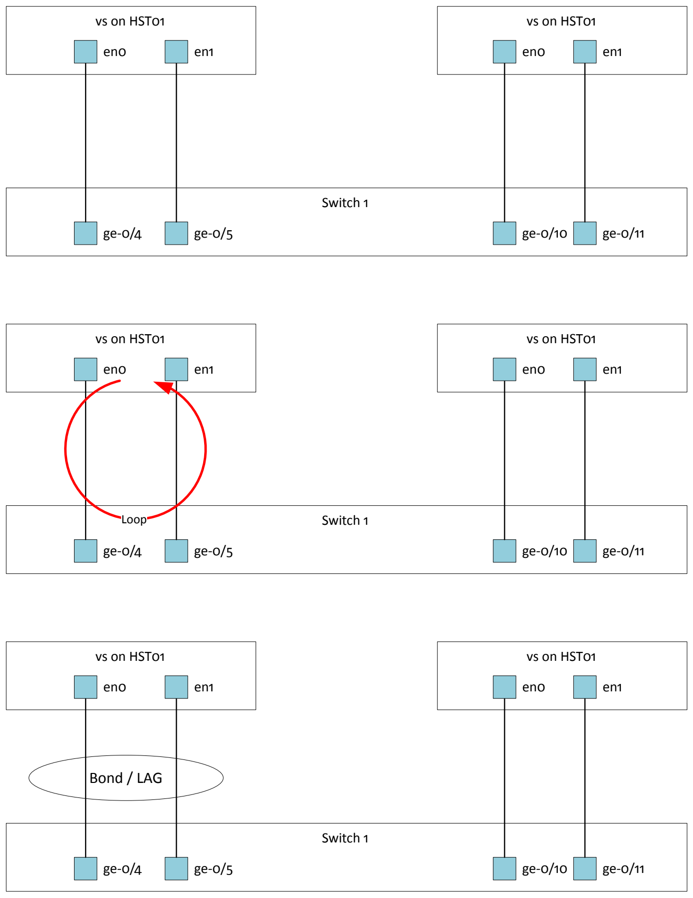

Figure 6: Switches with multiple uplinks

3.1 Switch dependant teaming

With switch dependant teaming the physical switches are aware that an interface is teamed together. This can be done using static configuration but that is not considered best practise. Instead Link Aggregation Control Protocol (LACP) is the standard for two switches to configure dynamic bonded interfaces.

For this to work all the ports on each end of the LAG need to be controlled by a single LACP process. Data centre design for resilience requires uplinks connect to different physical switches (or atleast two seperate switches). For LACP to work the switches need to be clustered in some fashion. This is often described as multichassis LAG (mLAG). This often requires the switches in the cluster to be similar or identical models. Another issue with using LACP (or static LAGs) is each LAG needs to be configured on the switch.

The advantage to using LAGs is many of the frame duplication corner cases are avoided. These will be examined further down.

3.2 Switch independant teaming

With the linux bond or the vswitch team an uplink can be either active or passive. A passive link will never transmit frames (but may received frames in some cases). Should all active links fail then any passive links will become active.

With multiple active uplinks the vswitch must decide which uplink to transmit frames on. By default this is done by hashing the source port ID of the frame and will only transmit down one uplink (see below of exception to this). Therefore for each frame there is effectively one active uplink and potentially multiple passive uplinks. This is important as it helps with duplicate frame suppression.

Normally for IPv4 (or IPv6) then a guest wants to communicate the first packet is an ARP (or Neighbour Discovery). This gets sent out as a broadcast, going down say en0 to switch 1. Switch 1 will forward this frame (as a broadcast) to switch 2, which will forward it back up en1 to the vswitch. This does not create a loop but does mean all guests will receive a second copy of the frame, unless it is suppressed. There are various methods for doing this but they also have corner cases, such as VIPs moving between guests or containers migrating across guests.

3.3 Split Physical Switches

vSwitches environments assume that every physical NIC is connected to every other NIC (ie fully meshed) but it is possible for things to still work should be be a network partition.

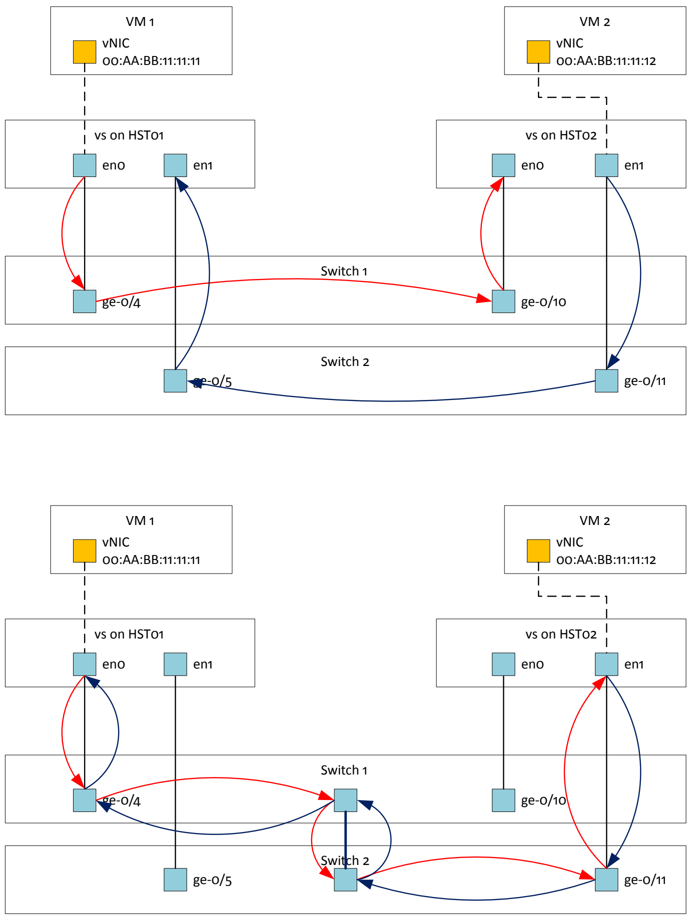

Figure 7: Switches with multiple uplinks

In the pathelogical case VM1 hashes out of en0 while VM1 hashes out of en1. Because s vswitch can receive frames on the passive link traffic get from VM1 to VM2 and in the converse. But because Switch 2 never sees VM1's MAC address it never learns it, so traffic from VM2 to VM1 is always treated as unknown unicast, and flooded to every port in the VLAN. Again the converse is true for VM1 to VM2 traffic.

When Switch 1 and Switch 2 are connected, the first frame (normally a broadcast ARP or ND but unknown unicast will work) is flooded to both ge-0/10 (up to en0 on HST02) and to Switch 2. Switch 2 then floods it to ge-0/11 (and up to en1 on HST02) so HST02 received a duplicate frame. However then return frame sent via en1 to Switch 2 is not flooded as Switch 2 has learn VM1 MAC address and will forward it only to Switch 1, which will in turn forward it to en0 on HST01.

3.4 Links failures in complex topologies

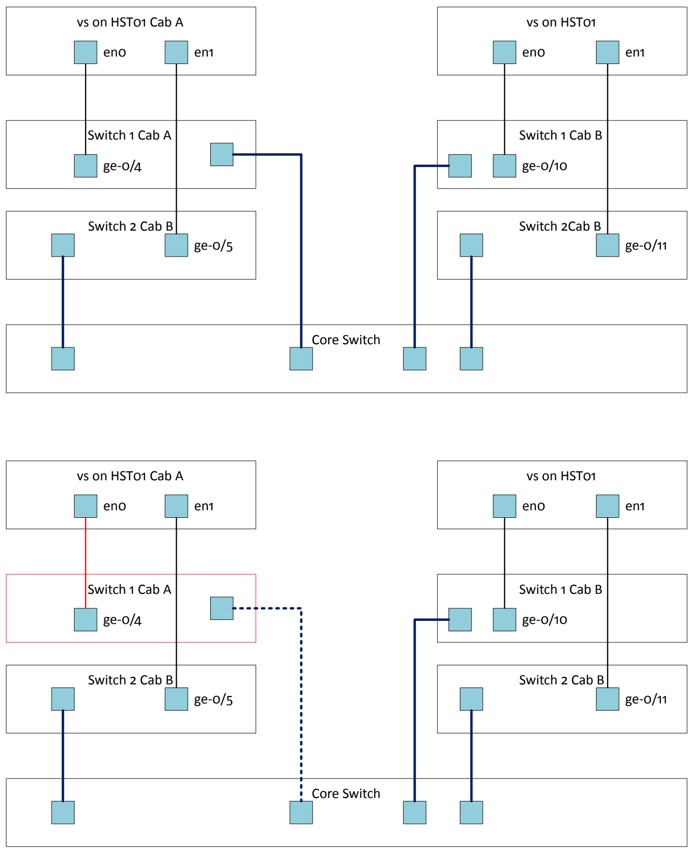

Figure 8: Switch topology with distant uplinks

In this (possibly contrieved) example two ESXi hosts are in different racks, with each rack having two Top Of Rack (TOR) switches. These switches are not connected to each other but instead all are connected to a central core switch (the Cisco Nexus FEX switches is an example of this).

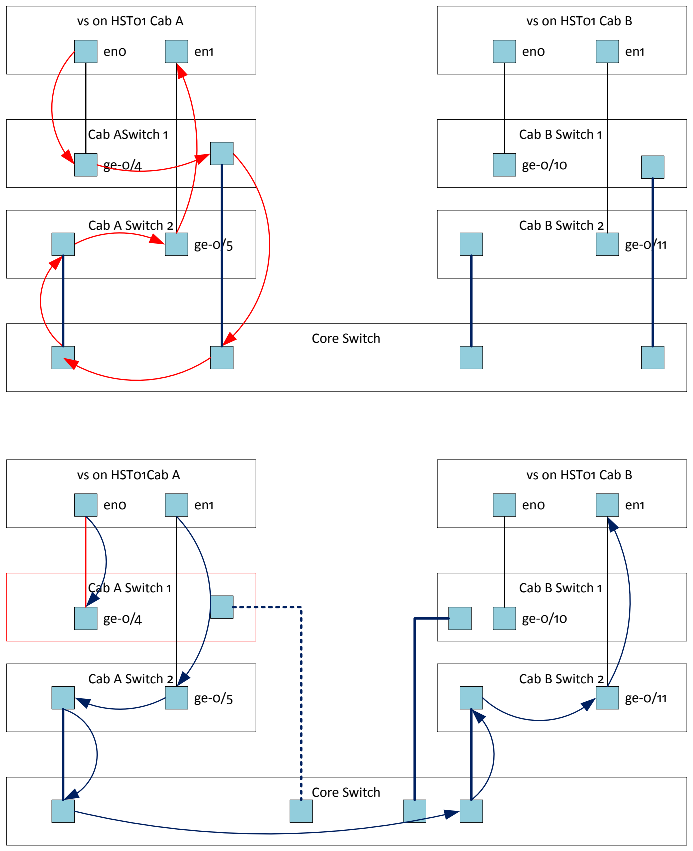

Figure 9: Switches with multiple uplinks

In the lower half of the figure Cab A Switch 1 has suffered an uplink failure (for example a cable fault). The vswitch will not detect this as its link to Switch 1 is still up, and therefore it will continue to send traffic to Switch 1, which will then blackhole the frames. In this case any traffic from guests hashing out of en0 on that host will disappear.

By default vswitches detect a link failure by using link state (which is to say is the NIC PHY receiving a signal). To try to detect upstream link failures vswitches can use Beacon Probing. This involves sending a broadcast frame (historically an RARP packet) down one uplink and listening on the other uplinks for it to return. This is shown in the top half of the figure.

With two uplinks (a very common configuration) should the probe not make it back the vswitch cannot determine which upstream switch has suffered a failure (cannot tell if the frame was lost on the way "out" or the way "back"). Therefore it uses a last ditch approach of sending every frame out of both uplinks, in the hope one is working. This is known as shotgunning the traffic (firing it out of both barrels). This can often be spotted by ping traffic showing duplicates, as the vswitches replicate frames.