Redbooks Paper

Vic Cross

Linux on IBM eserver zSeries and S/390: VSWITCH and VLAN Features of z/VM 4.4

Introduction and overview

z/VM® Virtual Switch (VSWITCH) is a new z/VM networking function announced by IBM® for z/VM Version 4 Release 4. This new feature is designed to improve the interaction between guests running under z/VM and the physical network connected to the zSeries® processor.

IBM also announced IEEE 802.1Q VLAN support for z/VM Virtual Switch, z/VM QDIO Guest LAN, and z/VM HiperSockets™ Guest LAN, allowing z/VM guests to create and participate in virtual LAN configurations.

This Redpaper describes how the features can be utilized by Linux guests, and how your virtual networking configurations can be greatly simplified through the use of these new functions.

The following topics are covered:

- Introduction to VLANs
- z/VM Virtual Switch (VSWITCH)
- Configuring Linux for VLAN
- Comparing VSWITCH with router-based designs
- High availability using z/VM Virtual Switch

We also describe some of our experiences with using z/VM Virtual Switch and VLANs on z/VM simulated networks.

Introduction to VLANs

Under z/VM 4.4, VSWITCH and Guest LAN include the capability to support IEEE 802.1Q Virtual LANs within the simulated network. This section introduces the concept of Virtual LANs and some of the terminology involved.

What is a Virtual LAN

A virtual LAN allows a physical network to be divided administratively into separate logical networks. In effect, these logical networks operate as if they are physically independent of each other.

The concept can be difficult to describe in words, so Figure 1 shows a diagram to illustrate it.

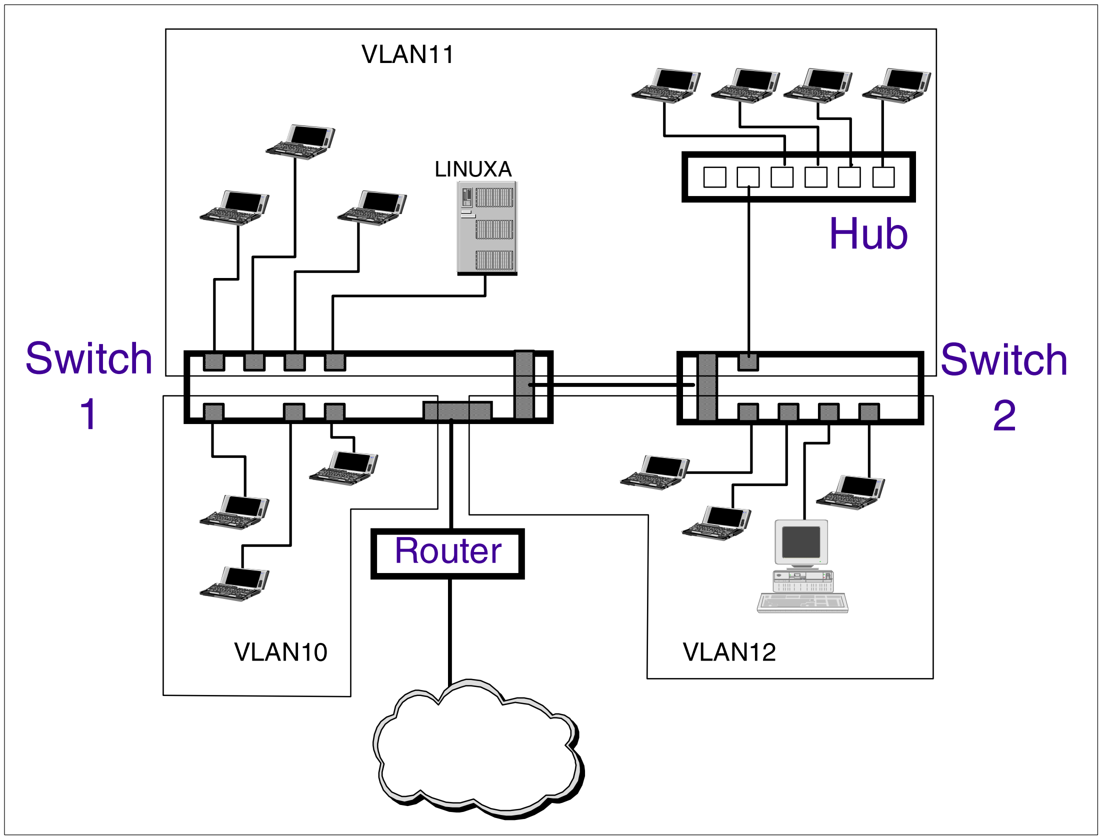


Figure 1 VLAN scenario

We make the following observations about the network in this diagram:

- Switch 1 is installed at one location, and Switch 2 and Hub are in another (for example, on separate floors of a building, or in separate buildings).
- Switch ports are represented by dark squares. The larger squares are *trunk* ports, the small squares are *access* ports.

Definitions: A *trunk* port is a port that carries VLAN traffic between VLAN-aware devices such as switches. Trunk ports provide the connections that carry traffic for more than one VLAN between the switches that are used to access those VLANs.

An *access* port is a port that is defined to a single VLAN only. Access ports provide connections for non-VLAN-aware devices, giving them access to the VLAN environment. Frames sent on access ports do not have any VLAN information attached.

- VLAN11 is a network that exists across both locations. This is achieved by defining trunk ports on both switches to the VLAN, and then connecting the two trunk ports.
- The router is a VLAN-capable device, attached to one of the trunk ports. The correct definitions in the router will provide a routing path between VLAN10 and VLAN12, and between either of these networks and the WAN (represented by the cloud). This is usually done by defining a virtual network device against the physical port on the router, linking that virtual interface to the VLAN, and enabling the interface for routing.
- VLAN11 has no access to any other VLAN, or the WAN. Even though VLAN11 shares access to trunk ports in both switches with VLAN12, the VLAN architecture prevents traffic from flowing between the two networks.

  If routing to VLAN11 is required, the Switch 1 trunk port to the router could be included into VLAN11 and a virtual interface for VLAN11 defined in the router.
- The only machines in the entire network that are permitted to access the server LINUXA are those in VLAN11. This is for the same reason that VLAN11 cannot access the external network: there is no routing path between VLAN11 and the other VLANs.

Attention: VLANs are different from Emulated LANs (ELANs). A VLAN uses the same frame format as the underlying medium, while an ELAN often uses one network technology to carry the frames of another. An example of this is Asynchronous Transfer Mode (ATM) ELAN, which allowed Ethernet and Token Ring traffic to be carried over an ATM backbone by emulating those frame formats over ATM cells.

Note that VLANs and ELANS should not be confused with the various simulated LAN technologies we use on zSeries and z/VM!

VLAN standards

Several virtual LAN mechanisms exist; most of them are proprietary and operate on a single vendor's equipment.

Port-based VLANs

Port-based VLANs are most often proprietary solutions that function within a single vendor's switch hardware. They provide a method of dividing a single network device (the switch) into separate broadcast domains; how the switch does this internally is platform-specific.

Importantly, end stations have no way to participate in multiple VLANs because the switch isolates the devices attached to it from the VLANs in the switch.

IEEE 802.1Q VLAN tagging

IEEE 802.1Q defines a standard virtual LAN mechanism that is being implemented across equipment from many different vendors. It uses a header, called the VLAN tag, added to packets transmitted over the network. The tag contains information that allows the network to manage the separation of the different virtual LANs.

Cisco Inter-Switch Link VLANs

Cisco Systems has a proprietary VLAN trunking mechanism called Inter-Switch Link (ISL). If your site uses Cisco networking equipment, ISL VLANs may be in use. ISL VLANs are not compatible with IEEE 802.1Q VLANs, but Cisco switches provide a function that allows mapping between ISL VLANs and 802.1Q VLANs.

Important: In this document we will investigate the IEEE 802.1Q VLAN only, because this is the VLAN standard supported under Linux and z/VM. For the remainder of this paper, when we refer to VLANs, we are specifically referring to IEEE 802.1Q VLANs (unless stated otherwise).

How IEEE 802.1Q VLANs work

VLAN adds additional information to the data packet. The extra data is referred to as the *VLAN tag*. The tag appears immediately after the Ethernet frame header, and before the data payload of the frame.

The format of the VLAN tag on Ethernet is shown in Figure 2.

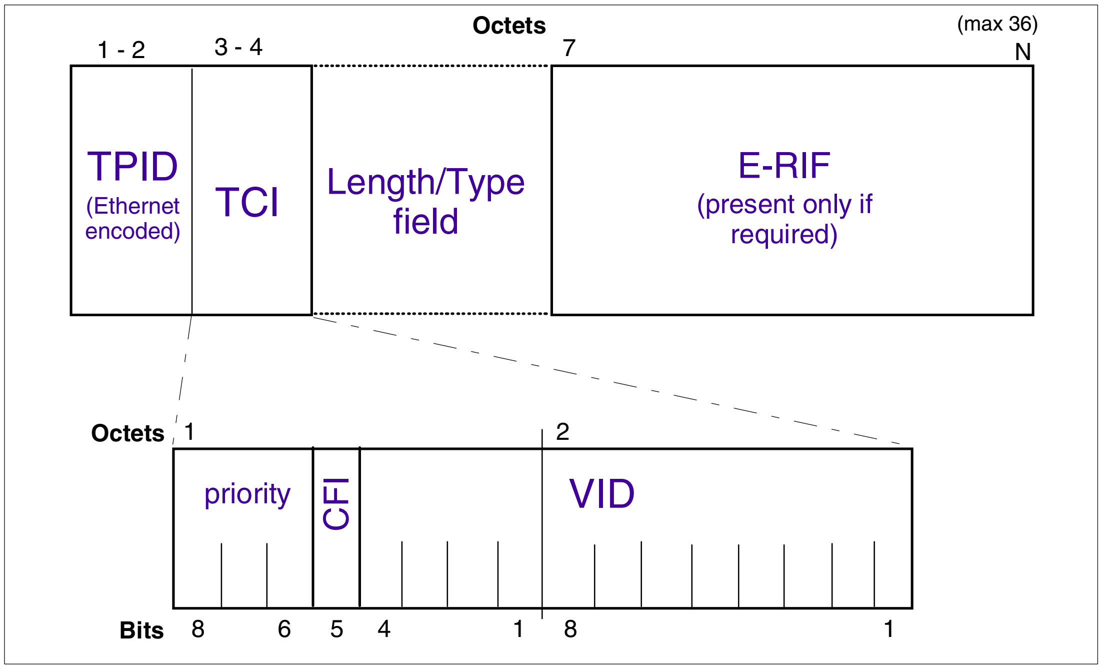


Figure 2 VLAN tag format for Ethernet

In most instances, only the Tag Protocol Identifier (TPID) and Tag Control Information (TCI) fields are used. This gives the impression that the VLAN header is only four bytes. In fact, the specification defines that additional information can be carried in the tag, including an Extended Routing Information Field (E-RIF) which would be used in source-routing environments (the Canonical Format Indicator (CFI) bit in the TCI indicates whether the E-RIF is present or not).

The three-bit priority field at the start of the TCI can be used by switch equipment to prioritize frames on different VLANs. On Linux, the `vconfig` command has parameters that allow the priority field to be set for the VLAN being defined.

Note: The VLAN priority is separate from IP priority mechanisms. VLAN priority is used to prioritize the frames of a VLAN relative to other VLANs, while IP prioritization operates within the IP layers of routers. Still other prioritization schemes may exist, like the traffic shaping facilities provided by Linux.

The VLAN tag is never included in a packet sent to a non-VLAN device. Part of the function of a VLAN-capable device is to add or remove VLAN tags as required, usually based on the learned capability of the peer device.

Untagged frames

A device does not have to support VLANs in order for it to join a network that uses them. A frame generated by a device that is not VLAN-capable is called an *untagged frame* when it arrives at the VLAN-capable switch.

The action taken by the switch in this case can vary. The switch can assign a default VLAN number to be assigned to any untagged frames that enter the switch, or it can tag the frames with a port-specific VLAN number (providing a function similar to a port-based VLAN).

Connecting to a VLAN

VLAN-capable devices attach to VLANs through the use of virtual network interfaces. Each VLAN has its own separate virtual interface. The diagram in Figure 3 shows the layered relationship of the components involved in packet transmission.

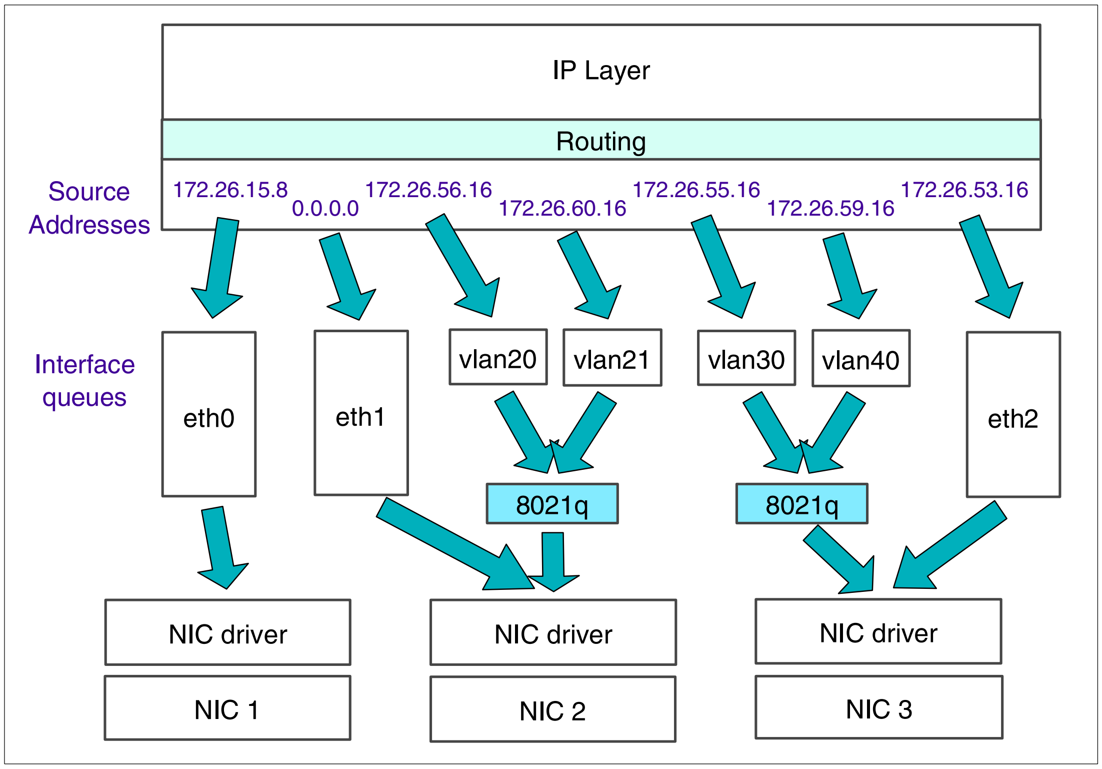


Figure 3 Layer diagram for VLAN-capable devices

Attention: This diagram is not meant to be a representation of the way that VLANs are implemented in any particular device. It is a conceptual overview only.

In Figure 3, *eth0* is a non-VLAN device. The IP layer, once a routing decision has selected eth0 as the interface for the packet to be transmitted on, places the packet onto the queue for eth0. Because eth0 is non-VLAN, when the packet gets to the front of the queue it is given to the NIC driver for encoding and transmission on the network with no other action. If the switch that this interface connects to is VLAN-aware, it must be configured to handle these frames; see “Untagged frames” on page 5 for more on this.

eth1 is an interface that supports connection to two VLANs, 20 and 21. The virtual interfaces for these VLANs are *vlan20* and *vlan21*, respectively. As for any other interface, packets may arrive at the queues for these interfaces as a result of the IP layer making a routing decision. For VLAN interfaces, however, the kernel VLAN code processes the packet by adding the VLAN tag and passing the tagged packet to the driver for the interface that the VLAN belongs to.

Note: In Linux, when VLAN interfaces are configured, the physical interface the VLAN is associated with must be provided as part of the configuration. This allows the VLAN support to send the packet to the correct interface queue.

The IP address associated with eth1 is a *dummy* address (that is, an address which does not match a real network configuration). Figure 3 shows eth1 configured as 0.0.0.0, which is commonly used for this purpose because it can never be used as a source address. The practical effect of this is that the IP interface eth1 will not generate any traffic, because the IP layer will not direct any packets to that interface for transmission (nor will the network send any packets to that address). This means that switch port does not need any configuration to support untagged frames (it can operate purely as a trunk port).

eth2 has an almost identical configuration to eth1, except for the address associated with it. Here, a valid IP address is configured on the eth2 interface. When the IP layer selects 172.26.53.16 as a source address, the packet is not processed by VLAN, resulting in an untagged frame being transmitted. The switch that eth2 is connected to must know what to do with such frames arriving when tagged frames are expected. The 802.1Q specification refers to this kind of port as a *hybrid port*, it can act both as a trunk port to handle tagged frames, and as an access port to handle untagged frames.

Note: Some equipment (such as Cisco switches/routers) specify a default VLAN used for any interfaces that do not have an explicit VLAN ID coded. This means that the device does not generate any untagged frames when communicating with VLAN-aware devices. Linux, on the other hand, can generate untagged frames in the way described for eth2.

The use of untagged frames in a VLAN environment can result in unexpected connectivity problems and potential security issues. To avoid this, always ensure that network traffic specifies a VLAN ID. You can do this by using a dummy address on the base interface, as illustrated with eth1 in Figure 3 on page 5. An example of what can happen with untagged frames in a VLAN environment is shown in “VLAN Isolation with untagged frames” on page 32.

VLAN support on z/VM Guest LAN

With z/VM 4.4, a simulated QDIO or HiperSockets network can now provide IEEE 802.1Q VLAN function to attached guests. The Guest LAN simulation works in exactly the same manner as in previous z/VM releases, except that it can pass VLAN tagged frames between guests that are VLAN-aware. When using VLANs over Guest LAN, the Guest LAN simulation does not interfere with the VLAN tagging of network frames. This means that a guest attached to a Guest LAN *must* be VLAN-aware in order to participate in the VLAN network.

Important: At this time, the HiperSockets microcode on zSeries processors does not support VLAN-tagged frames. VLAN support applies only to simulated HiperSockets on z/VM 4.4.

Another feature added to z/VM simulated HiperSockets with z/VM 4.4 is *broadcast capability*. This means that guests attached to a simulated HiperSockets can now use applications and protocols that use IP broadcast, in addition to the IP multicast support added in a previous release.

z/VM Virtual Switch (VSWITCH)

z/VM Virtual Switch is an extension of the Guest LAN simulated networking function provided in earlier releases of z/VM. VSWITCH provides a function that operates much like a Layer 2 switch within z/VM. To z/VM guests, it operates almost exactly the same as a z/VM 4.3 QDIO Guest LAN, with two important exceptions:

- Direct external network access via OSA Express
- IEEE 802.1Q VLAN support

We describe various functions provided by the new support in the following sections.

External network access using VSWITCH

VSWITCH provides a way to link an external network to guests under z/VM via an OSA Express, without the need for a routing function. Guests attached to the VSWITCH appear to be attached to the LAN that the OSA Express is attached to. This allows you to configure your guests with IP addresses from the network that the OSA Express connects to, without the need to configure Proxy ARP in a z/VM TCP/IP service machine.

Important: Avoid using the term “bridging” to describe this function. While this function looks like it bridges between a Guest LAN and the Ethernet, bridging usually refers to the copying of an entire Layer 2 frame from one network to another (performing whatever frame translation is required on the way, such as in the case of translational bridging between Token Ring and Ethernet). VSWITCH will handle only IP packets, and as such does not qualify as a full Layer 2 network bridge.

Because QDIO is an IP-only transport anyway, this is not a limitation. We mention this mainly so that you don’t incorrectly refer to VSWITCH as a bridge when talking to the network staff.

z/VM Virtual Switch can also function in a disconnected mode, where either an OSA port is not associated, or the associated OSA does not flow traffic to the external network. It might seem that a VSWITCH without an OSA is just the same as a QDIO Guest LAN, but this is not the case; the VSWITCH provides additional control over VLAN membership and handling of untagged frames (refer to “VLANs on z/VM Virtual Switch” on page 9 for more detail).

TCPIP service machine “controllers”

z/VM Virtual Switch uses a TCPIP service machine to act as the interface to an OSA Express network connection. This TCPIP service machine acts as a *controller* for the VSWITCH, and manages the operation of the OSA Express adapter ports the VSWITCH uses.

Important: The controller TCP/IP service machine is not involved in data transfer between the LAN and the guests on the VSWITCH.

In order for a VSWITCH to provide connectivity to a LAN, at least one TCPIP service machine must be configured to be a controller (this configuration is described in “Planning for VSWITCH” on page 10). The configuration allows the TCPIP stack to connect to the system service that manages VSWITCH connections. The TCPIP service machine is then able to act as a controller for a VSWITCH OSA connection.

Using the CP **QUERY CONTROLLER** command, you can see the TCPIP stacks that can serve as VSWITCH controllers. Example 1 shows the output from the command.

Example 1 CP QUERY CONTROLLER

```
q controller
CONTROLLER TCPCTL1   Available: YES   VDEV Range: *
  SYSTEM TESTSW3     Controller: *
CONTROLLER TCPCTL2   Available: YES   VDEV Range: *
Ready; T=0.01/0.01 17:12:16
```

When a VSWITCH with an OSA RDEV (real device) is defined, or when an OSA RDEV is added for an existing VSWITCH, the *VSWITCH system service determines which TCPIP service machine is to be used as the controller for that OSA. At least one controller TCPIP service machine must be eligible; otherwise, the request to add the OSA to the VSWITCH will fail.

If a CONTROLLER was specified on the CP **DEFINE VSWITCH** or CP **SET VSWITCH** command, and that TCPIP service machine is eligible to act as a controller, that stack will act as the controller. If no CONTROLLER parameter was given, or CONTROLLER * was specified, the *VSWITCH system service scans its list of eligible TCPIP service machines for the one with the fewest current VSWITCH connections and selects that service machine as the controller.

Once the controller stack for a VSWITCH has been selected, the *VSWITCH system service instructs it to activate the OSA port specified using the configuration given in the CP **DEFINE VSWITCH** or CP **SET VSWITCH** commands.

Important: You must not define the OSA Express port to your controller TCPIP service machine in its PROFILE TCPIP. The *VSWITCH system service dynamically adds the required definitions to the controller.

When a TCPIP service machine is controlling an OSA Express port for a VSWITCH, you can see the device and link entries created for the VSWITCH in the output of the **NETSTAT DEVLINKS** command. Example 2 shows the results of this command.

Example 2 NETSTAT DEVLINKS output on a TCPIP VSWITCH Controller

```
netstat devlinks
VM TCP/IP Netstat Level 440

Device VSWITCHDEV           Type: VSWITCH-IUCV Status: Connected
  Queue size: 0     CPU: 0  IUCVid: *VSWITCH   Priority: B
    Link VSWITCHLINK        Type: IUCV         Net number: 1
      BytesIn: 4355         BytesOut: 4489

Device TESTSW32320DEV       Type: VSWITCH-OSD  Status: Ready
  Queue size: 0     CPU: 0  Address: 2320      Port name: OSA2320
  Router Type: NonRouter    Arp Query Support: Yes
    Link TESTSW32320LINK    Type: QDIOETHERNET Net number: 0
      Broadcast Capability: Yes
      Multicast Capability: Yes
Ready; T=0.01/0.01 10:31:55
```

Here, you can see two extra devices added that support the VSWITCH:

- An IUCV link called VSWITCHDEV, created when the stack initializes with **`VSWITCH CONTROLLER ON`** specified in PROFILE TCPIP. This is the connection to the *VSWITCH system service. The link type shown here is VSWITCH-IUCV.
- The OSA Express port device, called in this case TESTSW32320DEV. This name is derived from the name of our VSWITCH (TESTSW3) and the virtual device number of the OSA.

Network data frames destined for guests attached to a VSWITCH are handled entirely by the CP VSWITCH buffer management logic. Frames are transferred directly between the OSA Express buffers and the guest on the VSWITCH. Likewise, outgoing frames are transferred by the VSWITCH code from the guest's virtual NIC to the OSA Express. The controller TCP/IP service machine is not involved in this data transfer.

Sharing an OSA Express port with z/VM Virtual Switch

The OSA Express port you use with your z/VM Virtual Switch can be shared between the VSWITCH and other guests (or LPARs), or even with other VSWITCHes. As long as you do not set PRIROUTER on your VSWITCH definition, you can share it with other images without restriction (even if one of the systems you are sharing with requires PRIROUTER).

You do not need to set PRIROUTER on a VSWITCH for normal operation. The only time to set PRIROUTER for a VSWITCH is if you have a guest attached to the VSWITCH that is providing a routing function for systems attached to another network.

There are additional considerations for OSA Express port sharing when multiple OSA Express ports are used for backup. See "Multiple OSA Express ports" on page 21 for more information.

VLANs on z/VM Virtual Switch

z/VM Virtual Switch supports IEEE 802.1Q VLANs. This means that guests attached to a VSWITCH under z/VM 4.4 can participate in VLAN networking. When an OSA Express is attached to the VSWITCH, VLAN operations extend between the VSWITCH and the LAN via the OSA Express.

On a VSWITCH, CP will act as a VLAN-aware switch and perform VLAN tag processing according to configuration. This means that the guest need not be VLAN-aware in order to communicate to a specific VLAN. In the case where the guest is linked to a VLAN via VSWITCH configuration, VSWITCH will add the correct VLAN tag to frames as they leave the guest and remove it when the guest receives frames. If the configuration of the VSWITCH does not specify a VLAN ID for that guest, the frames will be sent and received untagged.

Note: This means that the guest does not require any configuration for VLAN support. Guests that do not support VLAN will send and receive frames on the VLAN that CP has been configured to connect them to.

VSWITCH can provide many options for controlling which guests receive frames on which VLANs. This is called *VLAN filtering*, and is described in "VSWITCH VLAN filtering".

If an OSA Express is associated with your VSWITCH, this OSA Express port will function like a trunk port with access to all of the VLAN IDs appearing in the VSWITCH. When you configure the port on the LAN switch the OSA Express connects to, you can choose to make that port belong to any of the VLAN numbers configured in the VSWITCH. This will extend those VLANs out of the VSWITCH into the LAN; if the VLAN IDs already exist in the LAN, the guests on the VSWITCH will now be present on the same VLAN.

VLAN isolation

VLAN-capable devices automatically manage the separation of traffic between VLANs. Only a virtual adapter “attached” to the correct VLAN will receive packets on that VLAN.

VSWITCH VLAN filtering

VSWITCH provides an additional feature that provides further isolation. Using the CP **`SET VSWITCH`** command, you can control VLAN access at the VSWITCH. This is analogous to defining the VLAN membership of a trunk port on a LAN switch.

You can nominate the VLAN IDs that a guest is allowed to “see” when you use the CP **`SET VSWITCH`** command to grant access to a VSWITCH. Only frames tagged with those VLAN IDs are passed to the guest.

Attention: There appears to be one exception to this: regardless of the VLAN IDs specified, untagged frames from within the VSWITCH are always passed to the guest if the destination IP address matches an address that the guest has configured. This could result in unauthorized access to different network resources.

We recommend that you do not use untagged frames in your environment. For VLAN-unaware guests, use the CP **`SET VSWITCH`** command to grant the guest access to one VLAN ID only. For VLAN capable systems, always configure the base interface with a dummy IP address as described in “Connecting to a VLAN” on page 5.

The rules that apply to packet delivery on a z/VM Virtual Switch are explained in the “Working with Virtual Networks” chapter of *z/VM Virtual Machine Operation*, SC24-5955.

Using z/VM Virtual Switch

In this section we describe how we used VSWITCH on our system. We describe the planning and configuration steps we took to enable VLAN communication both within our z/VM system and to VLANs outside it.

Planning for VSWITCH

Before you can start using a VSWITCH, you must perform several tasks.

Important: The tasks described here are required only if you are using VSWITCH to connect to an external network via OSA Express. If you are using VSWITCH only as a Guest LAN with VLAN filtering, there are no preparation tasks and you can go straight to “Configuring VSWITCH” on page 11.

Update the directory entry for CONTROLLER TCPIPs

Any TCP/IP service machine that you want to operate as a controller for a VSWITCH must have the **IUCV *VSWITCH** statement added to its directory entry. This authorizes the TCP/IP service machine to connect to the *VSWITCH system service.

Add VSWITCH CONTROLLER to PROFILE TCPIP

For a z/VM TCP/IP service machine to initialize as a candidate to control an OSA for a VSWITCH, the **VSWITCH CONTROLLER** statement must appear in the PROFILE TCPIP for that stack.

Configuring VSWITCH

Once your z/VM system is prepared, you can start using VSWITCH.

Define the VSWITCH

A VSWITCH is created using the CP **`DEFINE VSWITCH`** command from a z/VM Class B userid. The full syntax of the command is explained in "DEFINE VSWITCH" on page 39. The key parameters (the ones you are most likely to use) of the command are as follows:

`switchname`	This is the name of the VSWITCH being created. This name is used to identify this VSWITCH in later commands.
`RDEV rdev`	The real device address of the OSA Express adapter to be associated with this VSWITCH. Only the first device address from the OSA port's address triple is entered here. You can specify up to three addresses at this parameter. This is used to provide backup OSA Express connections in case the first OSA fails. The special value NONE is used to create a Virtual Switch that is not associated with an OSA Express connection.
`PORTname portname`	The port name to be used when opening the OSA Express adapter. This works the same way as for normal use of the OSA Express: when sharing the OSA, all LPARs or guests must use the same name to open the OSA port. If you are specifying more than one RDEV, you must specify more than one portname.

Important: **`PORTNAME`** (and its operands) must be the last parameter specified on the **`DEFINE VSWITCH`** command (this applies to the **`SET VSWITCH`** command also).

The following shows the definition of a z/VM Virtual Switch called TESTSW3, with two OSA Express ports available.

```
define vswitch testsw3 rdev 2320 2180 portname osa2320 osa2180
VSWITCH SYSTEM TESTSW3 is created
Ready; T=0.01/0.01 20:13:02
HCPSWU2830I VSWITCH SYSTEM TESTSW3 status is ready.
HCPSWU2830I TCPCTL1 is VSWITCH controller.
```

Connecting a guest to VSWITCH

Guests connect to a VSWITCH using a simulated QDIO NIC. This works exactly the same way as it does for Guest LAN. There is an extra step involved with VSWITCH, however:

guests attaching to a VSWITCH must be granted access to the VSWITCH using the SET VSWITCH command.

Granting access to a VSWITCH using SET VSWITCH

Access to the VSWITCH is controlled by the GRANT and REVOKE options of the CP SET VSWITCH command. The GRANT option allows you to restrict the guest to particular VLANs, or allow the guest access to any VLAN (the default).

```
set vswitch testsw3 grant lnxsu6 vlan any
Command complete
Ready; T=0.01/0.01 14:38:01
set vswitch testsw3 grant lnxrh2 vlan 201
Command complete
Ready; T=0.01/0.01 14:38:10
```

The command syntax of the CP **SET VSWITCH** command is shown in “SET VSWITCH” on page 39.

Important: When a guest’s access to the VSWITCH is restricted to a single VLAN, untagged frames will still be delivered to that guest. Refer to “VSWITCH VLAN filtering” on page 10 for information on how VSWITCH controls frame flow.

Defining a simulated NIC

A simulated NIC is defined on a guest using the CP DEFINE NIC command.

Note: A simulated NIC of type QDIO can be connected to either a QDIO Guest LAN or a VSWITCH.

Refer to the section on creating a simulated NIC in *Linux on IBM @server zSeries and S/390: Large Scale Linux Deployment*, SG24-6824 for more information on how to define a simulated NIC. Also, an excellent new chapter called “Working with Virtual Networks” is available in *z/VM Version 4.4 Virtual Machine Operation*, SC24-6036.

This example shows the use of the CP **DEFINE NIC** command to define a simulated QDIO NIC at device address 6100.

```
define nic 6100 qdio
NIC 6100 is created; devices 6100-6102 defined
Ready; T=0.01/0.01 13:46:11
```

Attaching the simulated NIC to the VSWITCH

Once your simulated QDIO NIC is created, connect your simulated NIC using the CP **COUPLE** command. Here is an example of using the command to connect the simulated NIC at device address 6100 to a VSWITCH called TESTSW3.

```
couple 6100 to system testsw3
```

Configuring Linux for the VSWITCH connection

Once your userid has a simulated NIC defined and connected to the VSWITCH, you can IPL your Linux installation. Configuring your Linux system for attachment to a VSWITCH is done in the same way as for a QDIO Guest LAN connection.

Note: The IBM Redbook *Linux on IBM @server zSeries and S/390: Large Scale Linux Deployment*, SG24-6824 gives a description of how to set up Linux for Guest LAN attachment (in the section “Configuring a VM Guest LAN in a Linux guest”).

NICDEF directory control statement

NICDEF is a new directory control statement added to z/VM Directory Maintenance (DirMaint) with z/VM 4.4. It provides a new way to define simulated NICs as part of a guest’s directory definition. The options available on the DIRMAINT **`NICDEF`** command are shown in “DIRMAINT NICDEF” on page 40.

NICDEF provides some additional options not available when you use the **`SPECIAL`** directory control statement. Most relevant to Linux guests is the MACID parameter. This allows the MAC address allocated to a simulated NIC to be defined statically, rather than dynamically defined by CP when the NIC is created. This would be useful if you are using a protocol that relies on the MAC address of a host, such as DHCP. MACID specifies the last six hexadecimal digits of the MAC address; the first six are set using the new MACPREFIX option on the VMLAN statement in SYSTEM CONFIG.

Configuring Linux for VLAN

This section describes the process to start using VLANs in your Linux systems, whether on VSWITCH, HiperSockets Guest LANs, or OSA Express adapters that support VLAN. First we describe the steps that take place to start a VLAN interface, then we show you how you can do the steps automatically using the VLAN support included in SuSE SLES8.

VLAN configuration process on Linux

To connect to VLANs from a Linux system, you need the following:

- Linux kernel 2.4.14 or higher, or an earlier kernel with the IEEE 802.1Q VLAN patches.
- A user-space utility called vconfig, to configure the VLAN virtual interfaces.

Note: On our SuSE systems, not only was the kernel already prepared to support 802.1Q VLANs (the module 8021q.o was present), but the vconfig program was available in a prepared RPM package that came with the distribution (the vlan package).

The steps involved in starting a connection to a VLAN from a Linux guest are:

- Configure the VLAN.
- Load the IEEE 802.1Q VLAN module.
- Activate the physical interface.
- Configure a VLAN interface.
- Bring up the VLAN interface.

Configure the VLAN

If your VLAN configuration exists entirely within a VSWITCH or a z/VM simulated HiperSockets, there is no VLAN configuration required. On a VSWITCH, however, you do need to ensure that your guest has authority to use the VLAN you intend to use. Use the CP **`QUERY VSWITCH`** command to check VLAN authority, and the CP **`SET VSWITCH`** command to change the guest authorization.

Important: If the guest is already authorized to a VLAN and you need to change to a different VLAN, or you need to set or remove VLAN ANY authority, you can only change this by first revoking the current authority and granting the new authority.

If you will be connecting to a VLAN in your switch hardware, either directly or through a VSWITCH, ensure that the switch configuration supports the VLAN you intend to connect to. Refer to “Switch configuration” on page 24 for aspects of switch configuration that you may need to investigate.

Load the IEEE 802.1Q VLAN module

VLAN support is usually built as a module called 8021q.o. Use the **insmod** or **modprobe** command to load the module before attempting any operations to configure or use VLANs. The following shows how to load the module using the **modprobe** command:

```
# modprobe 8021q
```

You do not need to load a module if your kernel includes the VLAN support in the kernel (rather than in a module).

When the module is loaded, you will see the following messages in your system log or **dmesg** output:

```
802.1Q VLAN Support v1.7 Ben Greear <greearb@candelatech.com>
All bugs added by David S. Miller <davem@redhat.com>
```

Activate the physical interface

The physical interface used to connect to a VLAN must be activated prior to configuring any VLAN interfaces. If you are going to use a default VLAN or generate untagged frames, you can configure the interface in the same way as usual.

If you are not going to be using the physical interface to attach to an IP network (that is, the interface simply gives you a connection to the network, and al your IP addresses and network traffic will be associated with VLANs) you configure a dummy’ IP address on the physical interface. You can use the address 0.0.0.0 as a valid dummy address (in fact, the IP configuration scripts seemed to recognize what we were doing when we used 0.0.0.0, and our configuration was very easy).

On SuSE SLES 8, we used the ifcfg-ethX file in /etc/sysconfig/network to define the address for our base interface. Even when we used a dummy address, the network configuration scripts worked correctly.

Configure a VLAN interface

You create your VLAN virtual interface using the **vconfig** command. Apart from creating and deleting virtual interfaces, the vconfig command allows you to control the operation of the VLAN interfaces. You can specify the priority of packets on your VLANs, changing the relative priority of packets for different VLANs.

You can also change the default way in which the virtual devices are named. By default, the 8021q.o module uses the `DEV_PLUS_VID_NO_PAD` format. Some programs, however, have a problem with the period (.) character in the interface name, so either the `VLAN_PLUS_VID` or `VLAN_PLUS_VID_NO_PAD` formats must be used.

Important: One program that has a problem with the dot in the interface name is **`iptables`**, the interface to the netfilter code in the Linux kernel. The iptables program is used to set up firewall and network address functions in your Linux system.

If you plan to use any **`iptables`** commands against your VLAN interfaces, you will not be able to use the default name format.

Refer to “vconfig” on page 40 for an explanation of the **`vconfig`** command syntax. The following command will create a virtual interface for VLAN 10 against the eth0 interface:

```
# vconfig add eth0 10
```

If the default VLAN name format of `DEV_PLUS_VID_NO_PAD` is set, the command would create a VLAN virtual interface called `eth0.10`.

Tip: The vconfig usage text (shown in “vconfig” on page 40) shows you what your VLAN interface names will look like for different settings of the VLAN name format.

Bring up the VLAN interface

The VLAN interface added using vconfig supports the usual interface configuration commands such as **`ip`** and **`ifconfig`**. You configure the virtual interface in the same way as any other network interface on your system. The following command will configure the VLAN 10 virtual interface on eth0 with the IP address 192.168.10.29:

```
# ifconfig eth0.10 192.168.10.29 netmask 255.255.255.0 up
```

If the iproute2 utility is installed on your system, you can use the **`ip`** command to configure the interface instead. You would use these **`ip`** commands to get the same result as the **`ifconfig`** command above:

```
# ip addr add 192.168.10.29/24 dev eth0.10
# ip link set eth0.10 up
```

Tip: The iproute2 utility by Alexey Kuznetsov, which provides the **`ip`** command, allows a great degree of flexibility in configuring the Linux IP stack. Many distributors now include it by default, and have rewritten their network configuration scripts using iproute2 instead of the traditional IP configuration commands such as **`route`** and **`ifconfig`**. In addition, iproute2 provides the facilities to control Linux advanced routing features like multiple route tables, traffic shaping, and policy routing.

Configuring VLANs on SuSE SLES8

Since VLAN support is a fairly recent addition to Linux, configuring VLANs has been a manual process because the network configuration scripts provided with most distributions were not “VLAN-aware”. SuSE has set up their network scripts, however, so as long as you can use the right names for your VLAN interfaces, the system will set up your VLAN interfaces for you.

Restriction: In addition to the naming issue, the configuration requires that the vconfig per_kernel option be set to on. This means that VLAN numbers have scope across the kernel, rather than just per interface. If you need the ability to have the same VLAN numbers on different real interfaces representing different VLANs, you will not be able to use SuSE's configuration method and will have to create your own.

VLAN numbers should be allocated universally, so it is unlikely that this restriction will cause a problem for you.

Fixing the VLAN script

Unfortunately you may find a bug in the script used to configure the VLAN interface. In the /etc/sysconfig/network/scripts/ifup-802.1q script, the path to the vconfig executable may be incorrectly specified as /sbin/vconfig instead of /usr/sbin/vconfig. This will cause the script to fail.

Until SuSE has a fix for this, you will have to update the script to show the correct path to the vconfig program.

Configuring your VLANs

To configure a VLAN on SLES8, create a file in the /etc/sysconfig/network directory called ifcfg-vlanX, replacing X with the VLAN number you are configuring. The contents of this file should be similar to Example 3.

Important: The name of the ifcfg file you create is critical. If the file is named incorrectly (or more accurately: if the name you pass to the **`ifup`** script, which has to match the last part of the ifcfg file name, is not correct), the SuSE network configuration processing will not call the correct script to configure the interface.

As an example, if you are configuring VLAN ID 10, the details shown in Example 3 would be contained in a file called /etc/sysconfig/network/ifcfg-vlan10.

Example 3 ifcfg-vlan example

```
ETHERDEVICE='eth4'
BOOTPROTO='static'
STARTMODE='onboot'
IPADDR='192.168.201.109'
NETMASK='255.255.255.0'
NETWORK='192.168.201.0'
BROADCAST='192.168.201.255'
```

The ETHERDEVICE field is additional for VLAN configurations. This is used to instruct vconfig which physical interface to configure the VLAN on. The STARTMODE field works the same as for physical interfaces; this allows you to add your VLAN interfaces to your network configuration and have them activated automatically at boot-time.

Once you have prepared your ifcfg file, you start your VLAN interface by issuing the **`ifup`** command with the VLAN interface name as the parameter. The following command will start the vlan10 interface:

```
# ifup vlan10
```

Important: The physical interface (the interface specified in the ETHERDEVICE parameter in the interface configuration file) must be activated in order for this to function. See “Activate the physical interface” on page 14 for information on this.

Testing the configuration

Once your virtual interface is up, you can use it to communicate with other guests on the same VLAN. If your VSWITCH is linked through an OSA to stations on the LAN, and your LAN is configured with the same VLAN, you can now reach LAN devices on that VLAN as well.

Utilities

The **`ping`** command is a simple command you can use to verify connectivity. You can also use **`traceroute`**, a utility that will show you the path to other devices in the network.

Using traceroute, you can verify that your VLAN configuration is working as you expect. For example, two guests on the same VSWITCH but configured on separate VLANs can only reach each other via a router, and using traceroute will help you verify this. An example traceroute command execution is shown in Example 4.

Example 4   traceroute command example

```
# traceroute 192.168.200.109
traceroute to 192.168.200.109 (192.168.200.109), 30 hops max, 40 byte packets
 1  192.168.201.1  0.677 ms   0.805 ms   0.762 ms
 2  192.168.200.109  1.098 ms   1.026 ms   1.240 ms
```

Note: In “z/VM Virtual Switch failover” on page 28, we use the **`-R`** switch on the ping command to display the route taken both to and from the destination host. This can provide a little more information than traceroute. Refer to the discussion there for more details.

Making your VLAN configuration persistent

If you use the supplied configuration processes on SuSE SLES8 (discussed in “Configuring VLANs on SuSE SLES8” on page 15), you do not need to do anything else to have your VLAN connections start at reboot time. If you included “STARTMODE=onboot” in the `ifcfg-vlanX` file for your VLAN interface, the system network initialization scripts will do all the VLAN configuration for you at system startup.

If you are not using SuSE SLES8, or you cannot use SuSE’s process because of your configuration requirements, you will have to write a script that implements the steps outlined in “VLAN configuration process on Linux” on page 13.

Comparing VLANs on VSWITCH and Guest LAN

Configuring Linux guest connections to VLANs on a z/VM Guest LAN is identical to the process used on VSWITCH. The difference between Guest LAN and VSWITCH is in the handling of untagged frames.

In VSWITCH, if a guest authorized to connect to only one VLAN generates untagged frames, the VSWITCH tags them with the VLAN ID the guest is authorized to. In fact, the VSWITCH applies this VLAN tag to all the frames the guest generates, restricting that guest to generating frames for its authorized VLAN only[1].

The z/VM Guest LAN does not process or modify VLAN tags in any way. No VLAN filtering takes place. Guests that are VLAN-aware must be configured for matching VLAN IDs in order to communicate on their VLAN.

Frame processing for VLAN-aware and non-VLAN-aware guests occurs as follows:

- When guests that are VLAN-aware put tagged frames onto the Guest LAN, the Guest LAN will carry those frames unaltered and other guests that are VLAN-aware will be able to process them. Non-VLAN-aware guests will not be able to process these frames.
- Guests with no VLAN support generating untagged frames will not have their frames modified either, and both VLAN-aware and non-VLAN-aware guests will be able to process the packets (as long as the VLAN-aware guests have a network definition that supports untagged frames).

Comparing VSWITCH with router-based designs

Figure 4 shows a virtual network configuration based on Linux virtual routers. This configuration provides a high level of availability through its use of multiple router guests, multiple OSA ports, and dynamic routing facilities.

Note: The Redpaper *Linux on IBM @server zSeries and S/390: Virtual Router Redundancy Protocol on VM Guest LANs*, REDP3657, will give you more information about this configuration.

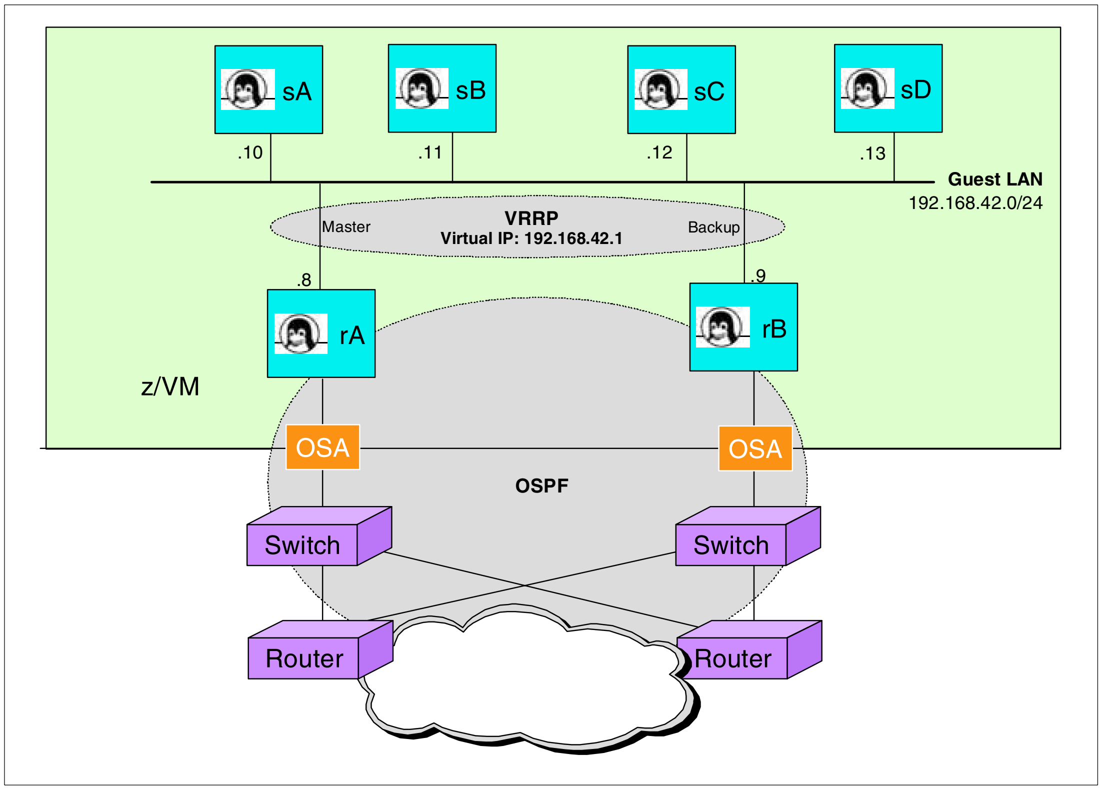


Figure 4 Connectivity based on Linux routers

[1] This is similar to the configuration of a “default VLAN” on some LAN switches.

While this configuration provides excellent availability for the Linux guests, it has a few drawbacks in a z/VM environment.

- OSA ports must be activated with the PRIROUTER setting. This stops you from sharing an OSA ports between multiple routers.
- The Linux guests consume memory, disk and CPU resource just by being active.
- Running the networking protocols that provide the routing failover (VRRP, OSPF) generates network traffic and consumes processor resource.

With VSWITCH, a highly available network configuration can be made without the resource overhead of virtual routers. Figure 5 shows an example configuration based on VSWITCH that provides a high degree of availability.

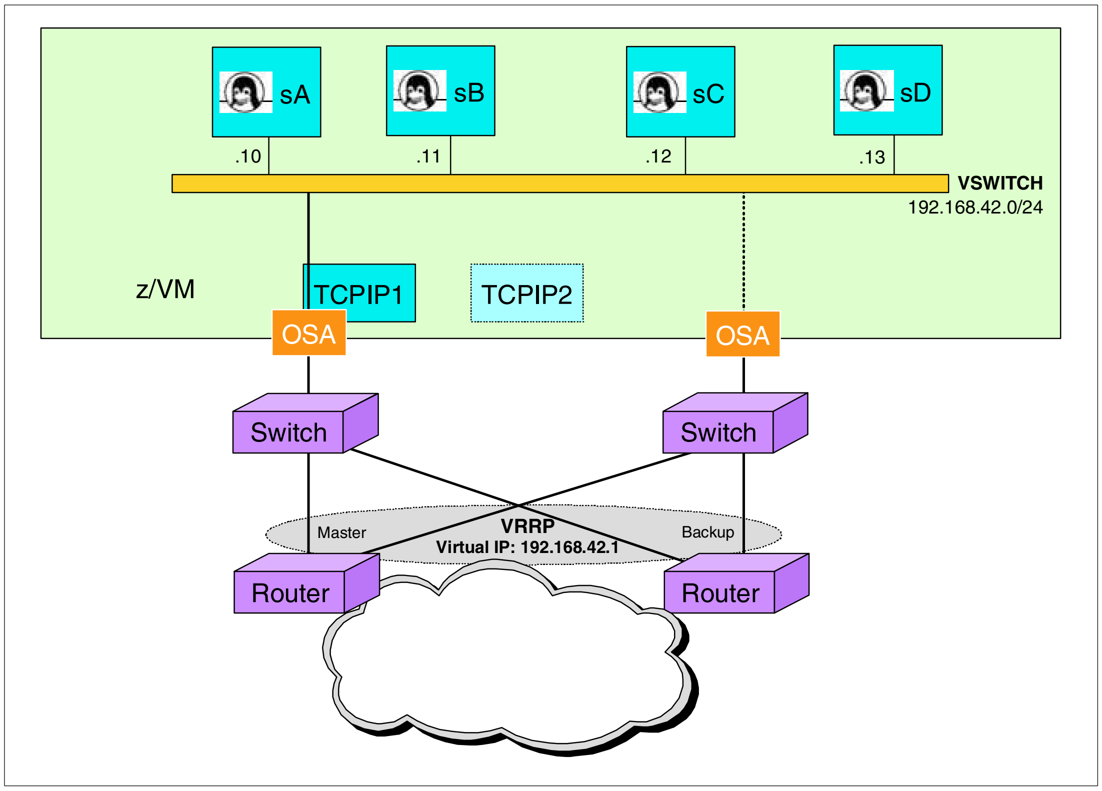


Figure 5   Connectivity based on VSWITCH

By using VSWITCH in this configuration, you remove the overhead of your Linux router guests, as well as the network traffic generated by the protocols used to provide high availability in the network. Importantly, the function of providing network availability is moved into network components, where it can be more consistently managed with the rest of the enterprise network function.

Note: In Figure 5, we show the two network routers participating in VRRP to provide a redundant gateway service for the Linux guests. While we strongly recommend that you use some kind of redundant gateway facility, it does not have to be VRRP. Other such protocols, like Cisco Hot Standby Router Protocol (HSRP), may be just as effective in your environment.

Also, not shown here is that the routers will be participating in the dynamic routing environment used for the rest of the LAN.

Migrating to z/VM Virtual Switch

Changing your current guest networking configuration over to z/VM Virtual Switch can be achieved with a minimum of reconfiguration effort.

QDIO Guest LAN to VSWITCH

If you are currently using persistent QDIO Guest LANs in your z/VM environment, you can simply replace them with a z/VM Virtual Switch. All your guests will connect to the VSWITCH in the same way as the Guest LAN they connect to now.

Restriction: If you use transient Guest LANs in your current configuration (LANs defined and owned by z/VM general users instead of SYSTEM), you will need to consider changing to persistent LANs. You cannot define a transient z/VM Virtual Switch; they are all owned by SYSTEM, and the CP commands that define and modify them can be run by Class B users only.

If your current Guest LAN is defined as UNRESTRICTED, you will need to add appropriate CP SET VSWITCH commands to give your guests authority to connect to the VSWITCH. If your current Guest LAN is RESTRICTED, you simply change your existing CP `SET LAN` commands into CP `SET VSWITCH` commands.

HiperSockets Guest LAN to z/VM Virtual Switch

This migration is a little more complex, because you cannot use the same simulated NICs for HiperSockets Guest LAN and VSWITCH. You will need to change the NIC definition for the guest from `TYPE HIPER` to `TYPE QDIO`. This change will require a corresponding change in the guest's TCP/IP configuration.

For this migration, we recommend setting up the new VSWITCH in parallel with the existing HiperSockets Guest LAN. Define the new simulated NIC to the guest, and complete the configuration of the new VSWITCH interface while the HiperSockets Guest LAN is still available. Alternatively (if you're comfortable with the Linux console), you can prepare new configuration files in advance, switch the guest's NIC definition to QDIO, and copy the new configuration files into place from the console.

vCTC or IUCV to z/VM Virtual Switch

Migrating either vCTC or IUCV topologies to z/VM Virtual Switch is a complicated process, because the topology of the guest connectivity changes from a set of point- to-point connections to a single shared-access transport facility.

Setting up the VSWITCH and simulated NICs in advance is definitely recommended for this migration.

Important: For *all* migrations to z/VM Virtual Switch, remember to verify that the guest's routing definitions are correct after the change is made. For example, if your VSWITCH connects to an OSA Express, the default route for the guests should specify a LAN router rather than a z/VM TCP/IP service machine or Linux router guest.

Virtual NIC definition

If you currently define your guests' Guest LAN connections using the `SPECIAL` directory control statement, consider using the new `NICDEF` statement instead. NICDEF provides additional parameters to define simulated NICs, including:

- Specifying the virtual channel path identifier (CHPID) the simulated NIC will use (most useful for simulated HiperSockets NICs for use on z/OS guests)
- Specifying the MAC address for the simulated NIC

NICDEF is discussed in "NICDEF directory control statement" on page 13. The syntax of the DIRMAINT NICDEF command is shown in "DIRMAINT NICDEF" on page 40.

High availability using z/VM Virtual Switch

z/VM Virtual Switch can be used to create highly-available connectivity for your Linux guests under z/VM. You can make access to your z/VM guests highly reliable by configuring the redundancy features of VSWITCH combined with LAN-based high availability.

You can define multiple OSA Express adapters for hardware redundancy, and multiple TCPIP controller service machines for some software redundancy. As long as your LAN switch is configured appropriately, you can ensure that your z/VM guests stay linked to the external network when failures occur.

Planning redundant configuration for VSWITCH

Some planning is required to ensure that your VSWITCH connectivity will survive outages.

Multiple CONTROLLER TCP/IP service machines

If you have only one TCPIP service machine and it is logged off or fails, all your external network connectivity via a VSWITCH fails also. For this reason, it is highly recommended to have at least one additional TCPIP service machine configured as a controller to provide continued connectivity.

When you have more than one TCP/IP service machine running in your z/VM system, these stacks can all be set up to become a VSWITCH controller. Using the CP `QUERY CONTROLLER` command, you can see the TCPIP stacks that can serve as VSWITCH controllers. Example 6 on page 24 shows the output from the command, and shows that our system currently has three TCPIP service machines ready to operate as controllers.

Tip: z/VM TCP/IP stacks used as controllers do not need to have any connectivity defined of their own. They do not need DEVICE and LINK statements, or HOME addresses, or the like. This means that you can set up additional TCPIP service machines with no specific IP configuration to serve as backup controllers for your VSWITCH configuration, with little configuration or management overhead. We describe a way of doing this in "Configuring controller TCP/IP service machines" on page 22.

Multiple OSA Express ports

When you specify more than one RDEV on the CP `DEFINE VSWITCH` command, redundant OSA ports are made available to your VSWITCH in case network access through the first port is impaired.

You can specify multiple OSA Express ports using extra values on the RDEV parameter to the CP `DEFINE VSWITCH` and CP `SET VSWITCH` commands. Up to three RDEV values can be provided, meaning up to three OSA ports can be provided to connect the VSWITCH to the LAN.

Important: Only one port is used for communication between the VSWITCH and the LAN at any one time. Additional ports defined for the VSWITCH are not connected until required. This means that your backup ports can be in use on other guests, or on other VSWITCHes.

However, if the PRIROUTER setting appears in the port configuration for both your VSWITCH and that of the guest or VSWITCH using the port, you will not be able to use that port as a backup (your VSWITCH will not be able to set PRIROUTER with the other connection active on the port).

A very important configuration requirement when defining multiple OSA Express ports for backup is that the switch configuration of the two ports must match. Importantly, if you are extending VLANs between your VSWITCH and the LAN, the switch ports you connect your OSA Express adapters to must all be defined as trunk ports, and they must all be configured to trunk the correct VLANs to the VSWITCH. If you don't do this, some or all of your guests may fail to connect properly if your main OSA Express connection fails.

Important: This may also prevent you from sharing an OSA Express port for backup purposes. If the primary system using the OSA Express port requires a certain configuration in the switch, and this conflicts with or does not match the configuration you need for the VSWITCH connection, the port cannot be used for VSWITCH backup.

Configuring high availability with VSWITCH

Figure 5 on page 19 shows an example of a high availability configuration. In the following paragraphs we will describe how to build it.

As discussed in "Planning redundant configuration for VSWITCH" on page 21, to make your VSWITCH connectivity redundant, you define multiple controller TCPIP service machines and provide multiple OSA Express ports to your VSWITCH.

In our lab configuration, we had two OSA Express ports at devices 2320-2322 and 2180-2182. The second OSA was the one used for direct access by other Linux guests to the LAN switch. We also defined two extra TCPIP service machines, named TCPCTL1 and TCPCTL2, to act as controllers.

Demonstration: "z/VM Virtual Switch failover" on page 28 shows the results of the testing we performed with these availability configurations.

Configuring controller TCP/IP service machines

You can use your main TCPIP service machine as a VSWITCH controller, but we decided to try setting up a separate configuration for a dedicated controller service machine. This configuration would allow us to easily and quickly start additional controller machines from the same configuration. The configuration we tried was a success, and the detail of how it was set up appears in the following paragraphs.

Minidisk configuration

The controller service machines were given a separate configuration disk, allocated as TCPMAINT 298 (the same size as TCPMAINT 198), which they link to as their 198 configuration disk. They each had their own 191 disk.

Setting up the configuration disk for the new service machines involved formatting the new minidisk and copying all the files from the configuration disk for the main TCP/IP service machine. We used the following commands:

```
FORMAT 298 K                # answer the prompts for volume label and confirm
COPYFILE * * D = = K (OLDDATE
```

Directory entries

We added the controller service machines to the z/VM directory as copies of the main TCPIP userid. The IUCV *VSWITCH directory entry is required to allow them to connect to the system service that manages VSWITCH connectivity to OSA Express adapters. Also, the link command for the machines' 198 disk must specify TCPMAINT 298 instead of TCPMAINT 198, as follows:

```
LINK TCPMAINT 0298 0198 RR
```

PROFILE TCPIP

The PROFILE TCPIP for our controller service machines requires no special customization, apart from the **`VSWITCH CONTROLLER ON`** statement required to allow the TCP/IP stack to initialize as a VSWITCH controller. We did not see a need in our configuration to restrict the machines to a range of LDEV (logical device) values; we used the default operation, allowing the OSA Express to be defined to the TCP/IP machine with the same logical device numbers as the real OSA device addresses (RDEVs).

Attention: While we did not need to specify the LDEV range in the **`VSWITCH CONTROLLER`** parameter in our configuration, you may need to do so to prevent the *VSWITCH system service from using device numbers that are already in use on your controller service machines. This may be particularly relevant if you are using your production TCP/IP service machine as a VSWITCH controller.

This configuration allowed us to have an identical PROFILE TCPIP on all the controller service machines and share the configuration disk between all the controller service machines we set up.

Important: If you specify the LDEV parameter, and it needs to be different on each of your controller service machines, you will need to have a separate 198 disk for each one. Allocate additional TCPMAINT minidisks as copies of the TCPMAINT 198 disk as described in "Minidisk configuration" on page 22 (using different device numbers for each, of course) and update each guest's directory entry to link to the correct minidisk.

SYSTEM DTCPARMS

Every TCP/IP service machine in your z/VM system needs to have DTCPARMS configured. You can add the definitions to the SYSTEM DTCPARMS file, or use a machine-local DTCPARMS file containing the definitions on a local minidisk to the guest. We chose the latter method, creating TCPCTL1 DTCPARMS and TCPCTL2 DTCPARMS files on the controller service machines' 198-disk.

Our DTCPARMS file was very simple, as shown in Example 5.

Example 5   TCPCTL1 DTCPARMS file

```
:NICK.TCPCTL1 :TYPE.SERVER   :CLASS.STACK
              :DISKWARN.NO
```

Start the controller service machines

Once configured, the new controller service machines can be logged on using the XAUTOLOG command.

Important: Remember to add your new service machines to your automation procedures to ensure that they start automatically after an IPL of z/VM.

Issue the CP **`QUERY CONTROLLER`** command to see that your service machines have initialized and that they have registered as controllers. Example 6 shows the results we obtained after our controller service machines initialized.

Example 6 CP QUERY CONTROLLER

```
q controller
CONTROLLER TCPCTL1   Available: YES   VDEV Range: *
  SYSTEM TESTSW3     Controller: *
CONTROLLER TCPCTL2   Available: YES   VDEV Range: *
Ready; T=0.01/0.01 17:12:16
```

Configuring multiple OSA Express ports

Remember, PORTNAME must be specified last on DEFINE or SET VSWITCH commands.

When we defined our z/VM Virtual Switch using the CP **`DEFINE VSWITCH`** command, we specified multiple values on the RDEV and PORTNAME parameters to provide access to our two OSA ports. Example 7 shows the results of our DEFINE VSWITCH command.

Example 7 DEFINE VSWITCH results

```
define vswitch testsw3 rdev 2320 2180 portname osa2320 osa2180
VSWITCH SYSTEM TESTSW3 is created
Ready; T=0.01/0.01 20:13:02
HCPSWU2830I VSWITCH SYSTEM TESTSW3 status is ready.
HCPSWU2830I TCPCTL1 is VSWITCH controller.
```

To get information about the z/VM Virtual Switch we just created, we can issue the CP **`QUERY VSWITCH`** command. Example 8 shows this output.

Example 8 QUERY VSWITCH results

```
query vswitch testsw3
VSWITCH SYSTEM TESTSW3  Type: VSWITCH  Active: 3     MAXCONN: INFINITE
  PERSISTENT  RESTRICTED    NONROUTER  MFS: 8192     ACCOUNTING: OFF
  State: Ready
  CONTROLLER: TCPCTL1       IPTIMEOUT: 5             QUEUESTORAGE: 8
  PORTNAME: OSA2320         RDEV: 2320 VDEV: 2320
  PORTNAME: OSA2180         RDEV: 2180
Ready; T=0.01/0.01 20:17:43
```

Switch configuration

To make sure you get the connectivity you expect in your environment, you will need to ensure that your switch configuration reflects the connectivity you want. Some of the things you need to look at or check are described here.

Note: Our testing environment used a Cisco Catalyst 6509 switch. We will illustrate the settings you need to investigate with configuration statements we used in our switch.

Setting up a trunk port

If you are using VLANs within your VSWITCH, and you want to link the VLANs inside z/VM with VLANs in the LAN, you will need to configure the switch port your OSA Express attaches to as a trunk port. This ensures two things:

- Frames sent to the OSA Express are VLAN tagged
- Frames for your desired VLANs are sent to the OSA Express

You can configure the trunk port to carry a specific range of VLANs. This would be useful to minimize unnecessary packet flow into the VSWITCH if you have many VLANs defined in the switch network but only some of them are represented in the VSWITCH.

On our Cisco switch, we defined a trunk port with this command:

```
set trunk 2/7 on dot1q 1-1005,1025-4094
```

This command sets port 7 in module 2 to be a trunk port, using IEEE 802.1Q tagging. VLAN numbers 1 through 1005 and 1025 through 4094 will be trunked on this port (this is the Cisco default).

Defining a VLAN

Your LAN switch must include a definition for the VLAN you want to create.

Important: When using VLANs in z/VM Guest LANs, you do not need to pre-define the VLANs in your z/VM configuration. If you are using VLAN filtering, however, you must grant access to VLAN IDs as required.

In Cisco CatOS, VLANs are defined using the **set vlan** command. The following command defined VLAN 202 on our switch.

```
itso6509 (enable) set vlan 202 type ethernet mtu 1492
```

Note: Depending on your logging level, some messages may appear when you enter a configuration command.

We specified an MTU of 1492 bytes to correspond with the MTU used on the OSA Express and VSWITCH adapters.

You can use the **show vlan** command to display attributes of your VLAN configuration. Example 9 gives an example of this.

Example 9 Displaying VLAN status on a Cisco Catalyst switch

```
itso6509 (enable) show vlan 202
VLAN Name                             Status    IfIndex Mod/Ports, Vlans
---- -------------------------------- --------- ------- ------------------------
202                                   active    114     2/3,2/7,2/11
                                                        15/1


VLAN Type  SAID       MTU   Parent RingNo BrdgNo Stp  BrdgMode Trans1 Trans2
---- ----- ---------- ----- ------ ------ ------ ---- -------- ------ ------
202  enet  100202     1492  -      -      -      -    -        0      0


VLAN MISTP-Inst DynCreated  RSPAN
---- ---------- ---------- --------
```

```
202  -          static     disabled


itso6509 (enable)
```

There are many other configuration settings for VLANs available. Refer to your Cisco documentation for more detail.

Adding ports to a VLAN

One of the first things you might want to do is configure certain ports on your switch to access certain VLANs. We used the CatOS **set vlan** command to assign switch ports to VLANs. The following command will attach port 3 on module 2 to VLAN 202.

```
itso6509 (enable) set vlan 202 2/3
```

The **show port** commands will display various information about your port status. Example 10 gives a sample of this display.

Example 10   Displaying port information on a Cisco Catalyst switch

```
itso6509 (enable) show port 2/3
* = Configured MAC Address

Port  Name                 Status     Vlan       Duplex Speed Type
----- -------------------- ---------- ---------- ------ ----- ------------
 2/3                       connected  202        a-full a-100 10/100BaseTX

<...>

Port  Security Violation Shutdown-Time Age-Time Max-Addr Trap     IfIndex
----- -------- --------- ------------- -------- -------- -------- -------
 2/3  disabled  shutdown             0        0        1 disabled      12

<...>

Port  Last-Time-Cleared
----- --------------------------
 2/3  Sun Jul 6 2003, 06:56:44

Idle Detection
--------------
   --
itso6509 (enable)
```

We have truncated some of the output of this command. This is shown by the <...> symbols in the output.

Routing to and from your VLAN

Stations attached to different VLANs cannot communicate to each other without a router. The router may be a physical device with attachments to ports on the switch, or it may be a software or hardware component of the switch.

The Cisco Catalyst switch in our lab was equipped with the Multilayer Switch Feature Card (MSFC), which provides a routing component. We used this routing function to provide access between our VLANs.

Configuring a connection to a VLAN for routing requires adding a virtual interface to the MSFC representing a connection to the VLAN. The **interface vlan** command is used to create a VLAN virtual interface, and the **ip address** command is used to assign an IP address to the virtual interface. Example 11 shows the commands we used to add a VLAN interface to the configuration of the routing function in our switch.

Example 11 Configuring MSFC for VLAN routing

```
itsores1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
itsores1(config)#interface vlan 202
itsores1(config-if)#ip address 192.168.202.1 255.255.255.0
itsores1(config-if)#^Z
itsores1#
```

Additional commands are required to enable routing in the MSFC, and to set up a dynamic routing protocol if you use one. Refer to Cisco documentation if you need assistance with these.

You can display the status of your VLAN virtual interface using the **show interface** command. Example 12 shows a sample of this.

Example 12 Interface status for a MSFC VLAN virtual interface

```
itsores1>show interface vlan202
Vlan202 is up, line protocol is up
  Hardware is Cat6k RP Virtual Ethernet, address is 0007.85f3.7302 (bia 0007.85f3.7302)
  Internet address is 192.168.202.1/24
  MTU 1500 bytes, BW 1000000 Kbit, DLY 10 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:01, output never, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue :0/40 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     556993 packets input, 28982480 bytes, 0 no buffer
     Received 556777 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     465 packets output, 38350 bytes, 0 underruns
     0 output errors, 2 interface resets
     0 output buffer failures, 0 output buffers swapped out
itsores1>
```

There are many other commands you can use to get information about your routing configuration. Refer to your Cisco documentation for more information.

Experiences

This section describes some aspects of VSWITCH and HiperSockets VLAN support that we encountered while writing this paper. The diagram in Figure 6 on page 28 shows our network testing configuration.

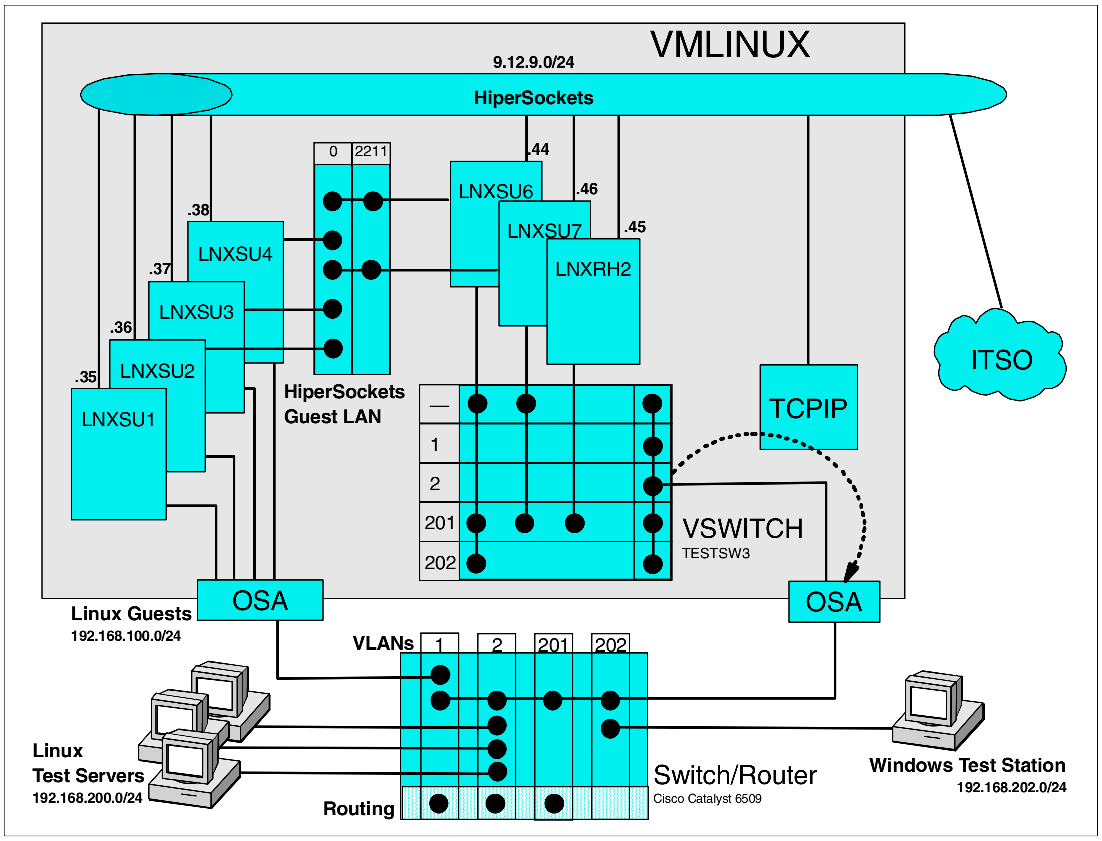


Figure 6 Test network configuration

In this diagram, we represented the VLANs in a switch (including z/VM Virtual Switch) as labelled rectangles within the representation of the switch. A particular station or guest's membership of a VLAN is indicated by a solid circle in the rectangle for the VLAN intersecting the connection from the station. In the VSWITCH, the ability to generate and receive untagged frames is shown by membership to the '—' VLAN.

For example, the guest LNXSU7 belongs to VLAN 201, but because we granted it access to VLAN ANY it can also generate and receive untagged frames. LNXRH2, on the other hand, was granted access to VLAN 201 only.

The diagram also shows a HiperSockets Guest LAN with a single VLAN, as well as the relationship between the VSWITCH OSA Express port and the controller TCPIP service machine.

z/VM Virtual Switch failover

With the configuration as described in "Configuring high availability with VSWITCH" on page 22, we tried various ways to disrupt the communication through the OSA Express port to the LAN.

We used the **`ping`** command from one of the Linux guest machines on our VSWITCH to a PC attached to a port on the LAN switch. We made sure that the LAN PC was only reachable via the VLAN link via the VSWITCH (that is, that there was no alternative path for the connection to travel over), and verified this by using the **`-R`** switch on the ping command.

Tip: Adding the **-R** switch on the ping command adds the RECORD_ROUTE option to the ICMP echo request packets that ping generates. Each host along the path adds its address to the IP header, so that when the reply is received by ping, it can display the route taken by the packet. The effect is similar to **traceroute**, but the response is much quicker and you obtain round-trip data (traceroute will only tell you the path from source to destination, and sometimes the return path is different).

Unfortunately, only nine hops can be counted using ping -R; if your round trip exceeds nine hops, you will only see the first nine.

Deactivate the switch port in the LAN switch

This failure is detected as a loss of signal or cable pull on the OSA Express, and the VSWITCH swaps to the other OSA port. We used the **set port disable** command in the Cisco Catalyst switch to shut down the switch port. Example 13 shows the results or our ping test.

Example 13 Ping test across OSA Express port detach

```
lnxsu6:~ # ping -R 192.168.200.100
PING 192.168.200.100 (192.168.200.100) from 192.168.200.109 : 56(124) bytes of data.
64 bytes from 192.168.200.100: icmp_seq=1 ttl=64 time=1.14 ms
RR:     192.168.200.109
        192.168.200.100
        192.168.200.100
        192.168.200.109

64 bytes from 192.168.200.100: icmp_seq=2 ttl=64 time=0.608 ms  (same route)
64 bytes from 192.168.200.100: icmp_seq=3 ttl=64 time=0.592 ms  (same route)
64 bytes from 192.168.200.100: icmp_seq=4 ttl=64 time=0.613 ms  (same route)
64 bytes from 192.168.200.100: icmp_seq=5 ttl=64 time=0.597 ms  (same route) < disable here
64 bytes from 192.168.200.100: icmp_seq=11 ttl=64 time=0.631 ms (same route)
64 bytes from 192.168.200.100: icmp_seq=12 ttl=64 time=0.624 ms (same route)
64 bytes from 192.168.200.100: icmp_seq=13 ttl=64 time=0.623 ms (same route)

--- 192.168.200.100 ping statistics ---
13 packets transmitted, 8 received, 38% loss, time 12006ms
rtt min/avg/max/mdev = 0.592/0.679/1.147/0.178 ms
lnxsu6:~ #
```

You can see that after a short period of time, the connection was transferred to the other OSA. On a z/VM console, we issued the commands and received messages as shown in Example 14.

Example 14 z/VM commands and messages around switch port deactivation

```
q vswitch testsw3                                                     <- before disable
VSWITCH SYSTEM TESTSW3  Type: VSWITCH  Active: 1     MAXCONN: INFINITE
  PERSISTENT  RESTRICTED    NONROUTER  MFS: 8192     ACCOUNTING: OFF
  State: Ready
  CONTROLLER: TCPCTL1       IPTIMEOUT: 5             QUEUESTORAGE: 8
  PORTNAME: OSA2180         RDEV: 2180
  PORTNAME: OSA2320         RDEV: 2320 VDEV: 2320
Ready; T=0.01/0.01 15:45:31
HCPSWU2830I VSWITCH SYSTEM TESTSW3 status is devices attached.       \    <- disable here
HCPSWU2830I TCPCTL1 is VSWITCH controller.                            \ unsolicited
HCPSWU2830I VSWITCH SYSTEM TESTSW3 status is ready.                   /   messages
HCPSWU2830I TCPCTL1 is VSWITCH controller.                           /
q vswitch testsw3
```

```
VSWITCH SYSTEM TESTSW3  Type: VSWITCH  Active: 1     MAXCONN: INFINITE
  PERSISTENT  RESTRICTED    NONROUTER  MFS: 8192     ACCOUNTING: OFF
  State: Ready
  CONTROLLER: TCPCTL1       IPTIMEOUT: 5             QUEUESTORAGE: 8
  PORTNAME: OSA2180         RDEV: 2180 VDEV: 2180
  PORTNAME: OSA2320         RDEV: 2320
Ready; T=0.01/0.01 15:48:11
```

In the console log of the TCPIP service machine, we saw some very good message output. This output is shown in Example 15.

Example 15   TCPIP console log output across switch port deactivation

```
15:47:52 DTCOSD309W Received adapter-initiated Stop Lan           1
15:47:52 DTCOSD082E VSWITCH-OSD shutting down:                    2
15:47:52 DTCPRI385I    Device TESTSW32320DEV:
15:47:52 DTCPRI386I       Type: VSWITCH-OSD, Status: Ready
15:47:52 DTCPRI387I       Envelope queue size: 0
15:47:52 DTCPRI388I       Address: 2320
15:47:52 DTCQDI001I QDIO device TESTSW32320DEV device number 2322:  3
15:47:52 DTCQDI007I   Disable for QDIO data transfers
OSA 2320 DETACHED TCPCTL1 2320 BY TCPCTL1                          4
OSA 2321 DETACHED TCPCTL1 2321 BY TCPCTL1
OSA 2322 DETACHED TCPCTL1 2322 BY TCPCTL1
15:47:52 DTCOSD080I VSWITCH-OSD initializing:                      5
15:47:52 DTCPRI385I  Device TESTSW32180DEV:
15:47:52 DTCPRI386I     Type: VSWITCH-OSD, Status: Not started
15:47:52 DTCPRI387I     Envelope queue size: 0
15:47:52 DTCPRI388I     Address: 2180
15:47:52 DTCQDI001I QDIO device TESTSW32180DEV device number 2182:  6
15:47:52 DTCQDI007I   Enabled for QDIO data transfers
15:47:52 DTCOSD238I ToOsd: Multicast support enabled for TESTSW32180DEV
15:47:52 DTCOSD319I ProcessSetArpCache: Supported for device TESTSW32180DEV
15:47:52 DTCOSD341I Obtained MAC address 0006296CA5BC for device TESTSW32180DEV
15:47:57 DTCOSD246I VSWITCH-OSD device TESTSW32180DEV: Assigned IPv4 address 192.168.200.109  7
15:47:57 DTCOSD246I VSWITCH-OSD device TESTSW32180DEV: Assigned IPv4 address 192.168.201.109
15:47:57 DTCOSD246I VSWITCH-OSD device TESTSW32180DEV: Assigned IPv4 address 192.168.202.109
```

The chain of events is as follows:

1. The OSA Express notifies TCP/IP that the LAN is no longer available.
2. TCP/IP commences a shutdown of the VSWITCH dynamically-defined device.
3. QDIO data transfer on the failed interface is terminated.
4. TCP/IP detaches the real OSA Express devices and attaches the devices for the backup OSA Express port (only the device detachments were logged).
5. TCP/IP initializes a new dynamically-defined device for the newly-attached OSA Express port.
6. QDIO data transfer for the new OSA Express port is activated.
7. The IP addresses for active guests are registered to the OSA Express port. LAN communications are re-established.

Detach the OSA from the controller service machine

The next thing we tried was to detach the real devices for the VSWITCH's active OSA Express port from the system. This allowed us to simulate the OSA Express device becoming unavailable through hardware failure or operator intervention.

The results were the same as for the switch port deactivation. Almost as soon as the disruption took place, the OSA Express detected a loss of communication and the VSWITCH was transferred to the other OSA Express port.

Attention: If you want to conduct similar testing, take care with your OSA connections. We found that repeatedly detaching the OSA Express devices from the TCPIP service machines eventually would cause problems with the OSA (channel errors were reported in the TCPIP log when the OSA was used, and communications would fail). To recover, we had to reset the OSA by configuring the channel path offline and back online.

Failure of the controller service machine

We simulated the failure of the controller service machine by using the CP FORCE command on its userid. Again we used ping to verify the communication path. The result of the ping test is shown at Example 16.

Example 16 Ping test across controller service machine outage

```
lnxsu6:~ # ping -R 192.168.200.100
PING 192.168.200.100 (192.168.200.100) from 192.168.200.109 : 56(124) bytes of data.
64 bytes from 192.168.200.100: icmp_seq=1 ttl=64 time=1.02 ms
RR:     192.168.200.109
        192.168.200.100
        192.168.200.100
        192.168.200.109

64 bytes from 192.168.200.100: icmp_seq=2 ttl=64 time=0.625 ms  (same route)
64 bytes from 192.168.200.100: icmp_seq=3 ttl=64 time=0.641 ms  (same route) <- force here
64 bytes from 192.168.200.100: icmp_seq=9 ttl=64 time=0.697 ms  (same route)
64 bytes from 192.168.200.100: icmp_seq=10 ttl=64 time=0.623 ms (same route)
64 bytes from 192.168.200.100: icmp_seq=11 ttl=64 time=0.621 ms (same route)

--- 192.168.200.100 ping statistics ---
11 packets transmitted, 6 received, 45% loss, time 10006ms
rtt min/avg/max/mdev = 0.621/0.705/1.025/0.146 ms
lnxsu6:~ #
```

You can see that communication was interrupted for a short time while another controller TCPIP stack took control of the VSWITCH. On a z/VM console, we issued the commands and received messages as shown in Example 17.

Example 17 z/VM commands and messages around controller failure

```
q vswitch testsw3
VSWITCH SYSTEM TESTSW3  Type: VSWITCH  Active: 1     MAXCONN: INFINITE
  PERSISTENT  RESTRICTED    NONROUTER  MFS: 8192     ACCOUNTING: OFF
  State: Ready
  CONTROLLER: TCPCTL1       IPTIMEOUT: 5             QUEUESTORAGE: 8
  PORTNAME: OSA2180         RDEV: 2180 VDEV: 2180
  PORTNAME: OSA2320         RDEV: 2320
Ready; T=0.01/0.01 15:48:11
HCPSWU2830I VSWITCH SYSTEM TESTSW3 status is controller not available. \   <- force here
HCPSWU2830I VSWITCH SYSTEM TESTSW3 status is ready.                     > unsolicited
HCPSWU2830I TCPCTL2 is VSWITCH controller.                             /    messages
q vswitch testsw3
VSWITCH SYSTEM TESTSW3  Type: VSWITCH  Active: 1     MAXCONN: INFINITE
  PERSISTENT  RESTRICTED    NONROUTER  MFS: 8192     ACCOUNTING: OFF
  State: Ready
  CONTROLLER: TCPCTL2       IPTIMEOUT: 5             QUEUESTORAGE: 8
  PORTNAME: OSA2180        RDEV: 2180 VDEV: 2180
  PORTNAME: OSA2320        RDEV: 2320
Ready; T=0.01/0.01 16:47:38
```

Important: We are very impressed with the way that z/VM Virtual Switch provides failover for its components; with only a short interval of dead time, the connectivity between the VSWITCH and the LAN is restored.

Ping is only a very simplistic test. Before you rely on this function to protect critical services, conduct your own testing to verify that the protocols and applications that you run in your environment recover from the brief outage interval.

VLAN Isolation with untagged frames

When setting up and using VLANs in our testing with VSWITCH, we were very satisfied with how the system preserved isolation of the different VLANs used. We wanted to verify what happened when untagged frames were used, especially with non-VLAN-aware guests.

To test this, we attached a guest running Red Hat Linux 7.2 to our VSWITCH (in Figure 6 on page 28, this was the guest LNXRH2). As distributed, Red Hat 7.2 has no support for VLANs. When we granted access to the VSWITCH, we granted it only to VLAN 201. Other than our Red Hat guest, there were two other guests on the VLAN, both VLAN-aware SLES8 systems.

We configured the Red Hat guest with an IP address configuration for that VLAN, and verified that we could communicate between it and other guests on the VLAN.

We then tried configuring different IP addresses as secondary addresses on the eth4 interface of the guest (the interface that connected to the VSWITCH).

Example 18   IP configuration on LNXRH2

```
[root@lnxrh2 root]# ip addr ls
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
2: eth4: <MULTICAST,NOARP,UP> mtu 1492 qdisc pfifo_fast qlen 100
    link/ether 02:00:00:00:00:10 brd ff:ff:ff:ff:ff:ff
    inet 192.168.201.10/24 brd 192.168.201.255 scope global eth4
    inet 192.168.100.10/24 scope global eth4
    inet 192.168.202.10/24 scope global eth4
3: eth2: <MULTICAST,NOARP> mtu 1492 qdisc noop qlen 100
    link/ether 02:00:00:00:00:0f brd ff:ff:ff:ff:ff:ff
4: hsi1: <MULTICAST,NOARP> mtu 8192 qdisc noop qlen 100
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
5: hsi0: <MULTICAST,NOARP,UP> mtu 16384 qdisc pfifo_fast qlen 100
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 9.12.9.45/24 brd 9.12.9.255 scope global hsi0
[root@lnxrh2 root]# ip route ls
192.168.100.0/24 dev eth4  proto kernel  scope link  src 192.168.100.10
9.12.9.0/24 dev hsi0  scope link
192.168.201.0/24 dev eth4  scope link
192.168.200.0/24 via 192.168.201.0 dev eth4
192.168.202.0/24 dev eth4  proto kernel  scope link  src 192.168.202.10
127.0.0.0/8 dev lo  scope link
default via 9.12.9.1 dev hsi0
[root@lnxrh2 root]#
```

Our Red Hat guest was authorized only to VLAN 201, corresponding to the 192.168.201.0/24 network. We should only be able to communicate directly with other guests on that same VLAN. By configuring IP addresses belonging to other VLANs against our interface, we tell the kernel to try and reach those networks directly instead of routing.

When we tried to ping the hosts on the 192.168.202.0/24 network, we got no responses. This confirmed that we were being isolated from VLAN 202. However, when we then tried to ping a host on the 192.168.100.0/24 network, we were surprised to find that we did get responses, even though we had restricted the guest to VLAN 201.

Example 19   Ping command on LNXRH2

```
[root@lnxrh2 root]# ping 192.168.100.109
PING 192.168.100.109 (192.168.100.109) from 192.168.100.10 : 56(84) bytes of data.
64 bytes from 192.168.100.109: icmp_seq=0 ttl=64 time=359 usec
64 bytes from 192.168.100.109: icmp_seq=1 ttl=64 time=292 usec
64 bytes from 192.168.100.109: icmp_seq=2 ttl=64 time=275 usec

--- 192.168.100.109 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.275/0.308/0.359/0.041 ms
[root@lnxrh2 root]#
```

To try and understand what was happening here, we had to look closer into the action of both the VSWITCH code and the Linux kernel.

The VSWITCH maintains a table of connected guests and associated IP addresses. The VSWITCH uses this table to determine which guest to send a received packet to. You can inspect this table using the CP QUERY VSWITCH command with the DETAIL option. Example 20 shows this command output after the ping command above.

Example 20   CP QUERY VSWITCH for IP address table

```
q vswitch testsw3 detail
VSWITCH SYSTEM TESTSW3  Type: VSWITCH  Active: 3     MAXCONN: INFINITE
  PERSISTENT  RESTRICTED    NONROUTER  MFS: 8192     ACCOUNTING: OFF
  State: Ready
  CONTROLLER: TCPCTL1       IPTIMEOUT: 5             QUEUESTORAGE: 8
  PORTNAME: OSA2320         RDEV: 2320 VDEV: 2320
  PORTNAME: OSA2180         RDEV: 2180
  RX Packets: 30569      Discarded: 33        Errors: 0
  TX Packets: 27494      Discarded: 0         Errors: 0
  RX Bytes: 2744464              TX Bytes: 2305386
    Authorized userids:
      LNXRH2   VLAN:  0201
      LNXSU6   VLAN:  ANY
      LNXSU7   VLAN:  0201   0202
      SYSTEM   VLAN:  ANY
    VSWITCH Connection:
      Device: 2322  Unit: 002   Role: DATA
      VLAN: ANY  Assigned by user
        Unicast IP Addresses:
          192.168.202.202  Mask: 0.0.0.0          Remote
    Adapter Owner: LNXRH2   NIC: 6100  Name: TESTSW3
      Device: 6102  Unit: 002   Role: DATA
      VLAN: 0201 Assigned by system
        Unicast IP Addresses:
          192.168.100.10   Mask: 255.255.255.0
          192.168.201.10   Mask: 255.255.255.0
          192.168.202.10   Mask: 255.255.255.0
        Multicast IP Addresses:
          224.0.0.1        MAC: 01-00-5E-00-00-01
    Adapter Owner: LNXSU6   NIC: 6100  Name: TESTSW3
      Device: 6102  Unit: 002   Role: DATA
      VLAN: ANY  Assigned by user
        Unicast IP Addresses:
          192.168.100.109  Mask: 255.255.255.0
          192.168.201.109  Mask: 255.255.255.0
          192.168.202.109  Mask: 255.255.255.0
          FE80::200:0:300:8/0                  Local
        Multicast IP Addresses:
          224.0.0.1        MAC: 01-00-5E-00-00-01
          FF02::1          MAC: 33-33-00-00-00-01 Local
          FF02::1:FF00:8   MAC: 33-33-FF-00-00-08 Local
Ready; T=0.01/0.01 15:22:41
```

The following chain of events takes place during our ping.

1. LNXRH2 sends an ICMP echo request packet. The source address is 192.168.100.10, because we have that address configured on one of our local interfaces and the route table selected it. The packet is given to the VSWITCH, and because LNXRH2 is authorized to only one VLAN ID the VSWITCH tags the frame for VLAN 201.
2. Next, the VSWITCH looks up the IP address table for the destination address. LNXSU6 has registered this address, so the next check is whether the VLAN authority of LNXSU6 will allow the frame to be delivered. LNXSU6 is authorized to VLAN ANY, so VSWITCH delivers the frame.
3. The IP stack in LNXSU6 does a final check to see if the destination address is local or if the packet has to be routed. The address is local, so the packet handled by the kernel.
4. The kernel processes the ICMP echo request, formatting an ICMP echo reply to be sent to 192.168.100.10. The route table selects 192.168.100.109 as the source address, and sends the reply via the eth4 interface. This packet will be sent as an untagged frame.
5. Because LNXSU6 is authorized to VLAN ANY, VSWITCH takes no action on the untagged frame entering the VLAN. It checks the IP address table again, this time for the destination address 192.168.100.10, and finds it on LNXRH2. Untagged frames are not filtered by VSWITCH, so the frame is sent to LNXRH2.
6. The IP stack in LNXRH2 checks to see if the destination IP address is local, which it is in this case, so the packet is given to its destination (the ping program).

The main reason this connection can take place is an efficiency aspect of the Linux kernel. At step 3, when the packet arrives at the VLAN interface, it technically has an incorrect destination address for the interface it is arriving on. Rather than forwarding the packet through the stack to deliver it at the correct interface, the kernel will simply accept it at the interface it was delivered.

Many IP stacks work this way to improve performance.

The other reason that the connection works is that the VSWITCH does not filter untagged frames for delivery to guests.

In this instance, since the two machines were configured with a common network interface and were able to communicate anyway, the fact this happens this way is a curiosity. However if the machines were not intended to have a shared network then this could lead to unexpected connectivity and a loss of security.

Closing the door

There are at least two ways you can eliminate this possible exposure: by eliminating untagged frames, or by using firewall rules in Linux. In the following sections, we discuss these methods in more detail.

Eliminate untagged frames

In the configuration of LNXSU6, the base interface (eth4) was used to gain connectivity to the other Linux guests that were attached to the other OSA device in the network (the 192.168.100.0/24 network). The Cisco switch that we connected to was defined to use VLAN 1 as a default VLAN ID, so all the Linux guests attached to the other OSA were actually in VLAN 1, as far as the switch was concerned.

By changing the configuration of LNXSU6 to bring up a connection to the 192.168.100.0/24 network via VLAN 1 and using a dummy IP address on the base interface, we could still communicate with the other machines on the 192.168.100.0/24 network. Importantly, this configuration eliminated untagged frames from LNXSU6. The frames sent back to LNXRH2 were not delivered by the VSWITCH because they were tagged as VLAN 1, a VLAN that LNXRH2 was not authorized to.

Firewall rules in Linux

Many administrators do not like the default behavior of Linux accepting packets on “foreign” interfaces. By adding iptables rules in any Linux host attached to multiple VLANs, you can prevent the kernel from accepting packets on an interface whose network does not match the destination address of the packet.

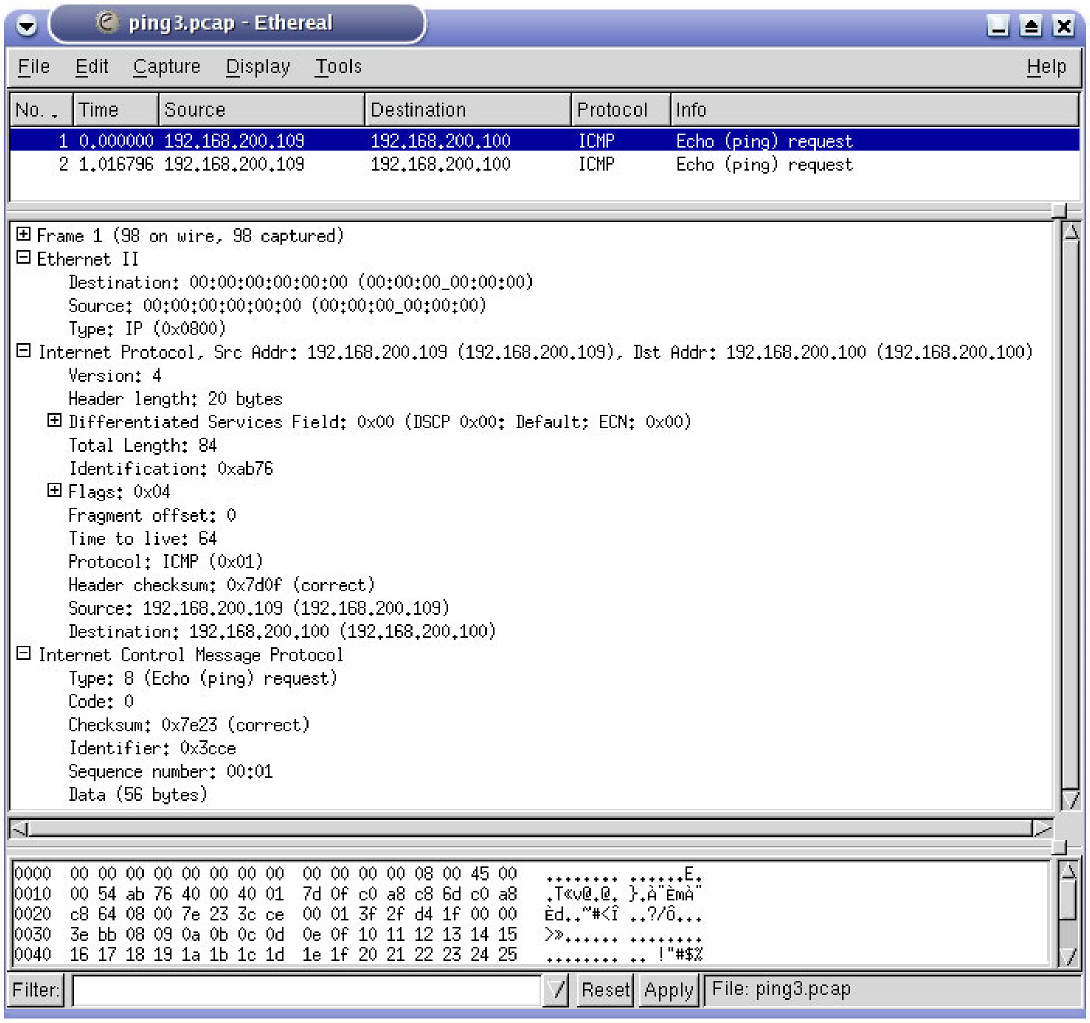
Attention: We are not trying to point out a design problem by illustrating this issue; we do not believe that a design problem exists. Rather, we are drawing your attention to an interoperation aspect that you should be aware of when implementing VSWITCH and VLAN in a Linux environment.

Capturing VLAN network traffic on VSWITCH

We used the tcpdump packet capture tool to grab network traffic, and the Ethereal analysis tool to look at the trace.

Attention: To use tcpdump with QETH network interfaces, the tcpdump-qeth wrapper script by Holger Smolinski of IBM is needed to analyze the frames correctly. This script is included in the tcpdump package on SuSE SLES 8.

We started a packet capture against the VLAN 2 interface (the 192.168.200.0/24 network), and issued ping commands for destinations on the various VLANs that our host was attached to. Analyzing the captured packets, we only saw the packets on VLAN2. This was what we expected. Figure 7 on page 36 is a screen capture of the Ethereal program analyzing our captured traffic.


ping3.pcap - Ethereal
File Edit Capture Display Tools Help
Filter: Reset Apply File: ping3.pcap

```
No.  Time      Source           Destination      Protocol  Info
  1  0.000000  192.168.200.109  192.168.200.100  ICMP      Echo (ping) request
  2  1.016796  192.168.200.109  192.168.200.100  ICMP      Echo (ping) request
```

```
⊞ Frame 1 (98 on wire, 98 captured)
⊟ Ethernet II
    Destination: 00:00:00:00:00:00 (00:00:00_00:00:00)
    Source: 00:00:00:00:00:00 (00:00:00_00:00:00)
    Type: IP (0x0800)
⊟ Internet Protocol, Src Addr: 192.168.200.109 (192.168.200.109), Dst Addr: 192.168.200.100 (192.168.200.100)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 84
    Identification: 0xab76
  ⊞ Flags: 0x04
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (0x01)
    Header checksum: 0x7d0f (correct)
    Source: 192.168.200.109 (192.168.200.109)
    Destination: 192.168.200.100 (192.168.200.100)
⊟ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x7e23 (correct)
    Identifier: 0x3cce
    Sequence number: 00:01
    Data (56 bytes)
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 54 ab 76 40 00 40 01  7d 0f c0 a8 c8 6d c0 a8   .T«v@.@. }.À¨ÈmÀ¨
0020  c8 64 08 00 7e 23 3c ce  00 01 3f 2f d4 1f 00 00   Èd..~#<Î ..?/Ô...
0030  3e bb 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   >»...... ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
```

Figure 7   Ethereal packet analysis on VLAN 2

We highlighted one packet for display in the decode window.

Note: In the Ethereal display shown in Figure 7, you can see that the Ethernet II header has all-zero MAC addresses for source and destination MAC. This is the dummy Ethernet header inserted into the frame by the tcpdump-qeth script.

You can also see that we only appear to have captured the transmitted packets. It appears that the same “feature” that we discussed in “VLAN Isolation with untagged frames” on page 32, which is Linux receiving packets on any configured interface as long as the IP address is local, is apparent here.

In this case, it would seem that the frames are being delivered to the base interface and accepted there rather than registering against the VLAN interface. Looking at the statistics for the VLAN 2 interface, this definitely appears to be the case (refer to Example 21).

Example 21   Interface statistics for a VLAN interface

```
lnxsu6:~ # ifconfig vlan2
vlan2     Link encap:Ethernet  HWaddr 02:00:00:00:00:08
          inet addr:192.168.200.109  Mask:255.255.255.0
          inet6 addr: fe80::200:0:300:8/10 Scope:Link
          UP RUNNING MULTICAST  MTU:1492  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7783 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:664403 (648.8 Kb)
```

You can see that no received packets have been recorded against this interface.

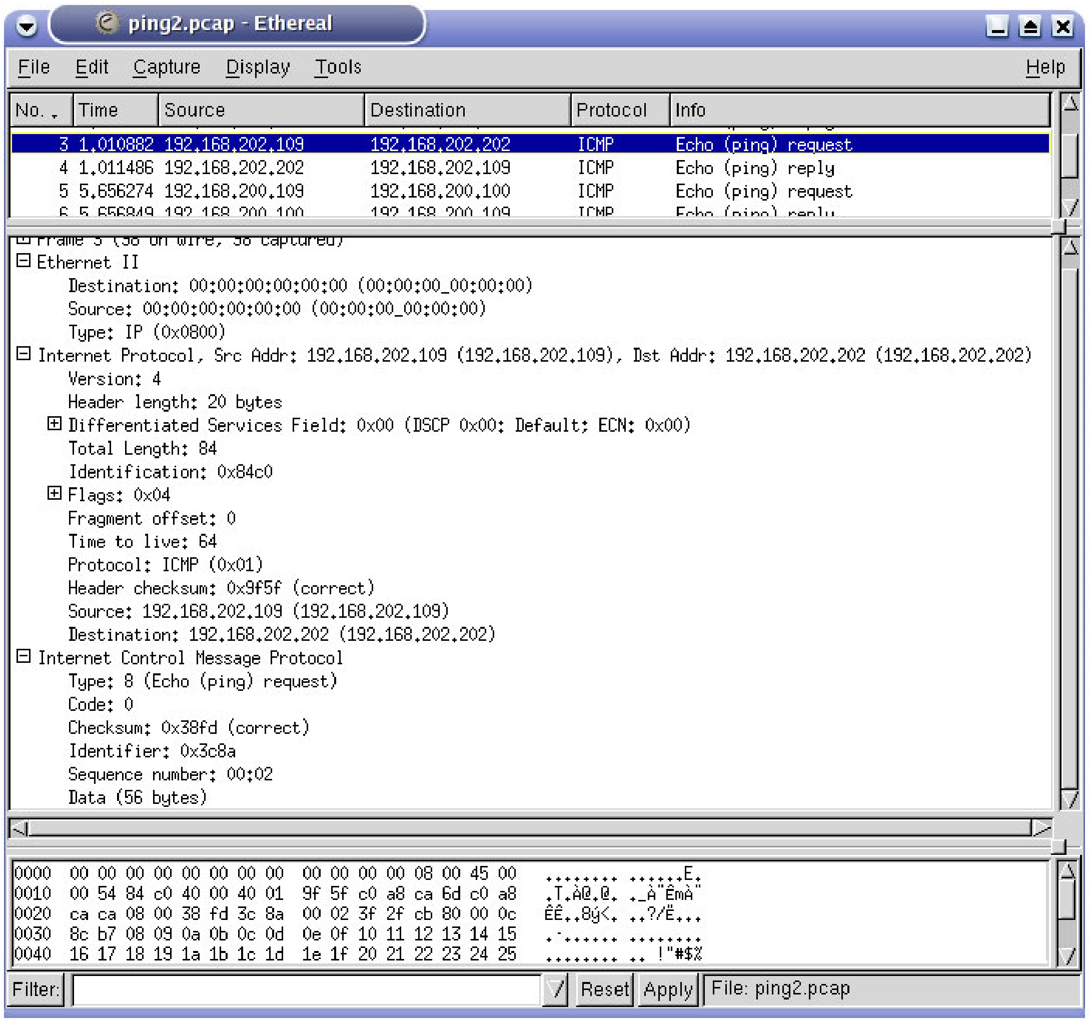
Seeing this behavior, we then traced the base interface. In doing so we hoped to see frames with VLAN headers in the trace. We wanted to see the contents of the VLAN headers and other changes to packets that the VLAN support made.

Two things surprised us about what we saw in the trace of the base interface:

- We saw no VLAN headers
- We saw both transmitted and received IP packets for all VLANs

Figure 8 on page 38 shows Ethereal analyzing a packet trace we collected against the eth4 interface on LNXSU6. In particular, we ran tcpdump with the **-p** option, which prevents tcpdump from using promiscuous mode (we wanted to be absolutely sure we were not opening anything up that we did not intend to[2]).

[2] Promiscuous mode gave the same result, which shows it is fairly meaningless on z/VM simulated networks anyway.



```
No.  Time     Source           Destination      Protocol Info
  3 1.010882 192.168.202.109  192.168.202.202  ICMP     Echo (ping) request
  4 1.011486 192.168.202.202  192.168.202.109  ICMP     Echo (ping) reply
  5 5.656274 192.168.200.109  192.168.200.100  ICMP     Echo (ping) request
  6 5.656849 192.168.200.100  192.168.200.109  ICMP     Echo (ping) reply
```

```
⊞ Frame 3 (98 on wire, 98 captured)
⊟ Ethernet II
    Destination: 00:00:00:00:00:00 (00:00:00_00:00:00)
    Source: 00:00:00:00:00:00 (00:00:00_00:00:00)
    Type: IP (0x0800)
⊟ Internet Protocol, Src Addr: 192.168.202.109 (192.168.202.109), Dst Addr: 192.168.202.202 (192.168.202.202)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 84
    Identification: 0x84c0
  ⊞ Flags: 0x04
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (0x01)
    Header checksum: 0x9f5f (correct)
    Source: 192.168.202.109 (192.168.202.109)
    Destination: 192.168.202.202 (192.168.202.202)
⊟ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x38fd (correct)
    Identifier: 0x3c8a
    Sequence number: 00:02
    Data (56 bytes)
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 54 84 c0 40 00 40 01  9f 5f c0 a8 ca 6d c0 a8   .T.À@.@. ._À¨ÊmÀ¨
0020  ca ca 08 00 38 fd 3c 8a  00 02 3f 2f cb 80 00 0c   ÊÊ..8ý<. ..?/Ë...
0030  8c b7 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   .·...... ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
```

Figure 8   Ethereal packet analysis on base interface

In the top portion of the window, you can see some of the packets that were collected. The source and destination addresses for these packets are for addresses on VLAN 2 and VLAN 202 in our configuration.

It appears that tcpdump can only see IP frames when QETH is used. Since QETH is an IP-only transport, this makes sense. By the time tcpdump views the packets, the 802.1Q processing has finished with them, having extracted the VLAN headers and leaving just the IP payload.

Note: With the IBM s390 kernel patches for the 2.4.21 kernel, the QETH driver has been released as open source. An interesting project would be to modify the driver so that "traditional" tools like libpcap function as expected. This may allow better visibility not only of VLAN frames, but of other frame headers as well.

However, this still may not produce the desired results because the Ethernet header is not always preserved by the OSA Express hardware. Also, not all frames received on a QETH interface will ever have had an Ethernet header: traffic between guests on the same simulated network, for example, do not.

Command usage

The usage syntax of the commands used in this paper are shown in this section.

DEFINE VSWITCH

The syntax of the DEFINE VSWITCH command is shown in Example 22.

Example 22 DEFINE VSWITCH syntax

```
DEFINE VSWITCH

                                 (1) .-RDEV--NONE--------.  .-CONnect----.
 >>--DEFINE VSWITCH--switchname------+-------------------+--+------------+---->
                                     |       .---------. |  '-DISCONnect-'
                                     |       V      (2)| |
                                     '-RDEV----rdev----+-'

    .-QUEuestorage--8M------.  .-CONTRoller--*-------.  .-IPTimeout--5---.
 >--+-----------------------+--+---------------------+--+----------------+---->
    '-QUEuestorage--numberM-'  '-CONTRoller--userid1-'  '-IPTimeout--nnn-'

    .-NONrouter-.
 >--+-----------+--.---------------------------.-----------------------------><
    '-PRIrouter-'  |           .-------------. |
                   |           V          (3)| |
                   '-PORTname----portname----+-'

 Notes:
 (1)  You can specify the operands in any order, as long as switchname is the
      first operand specified, and portname is the last operand specified, if
      applicable.

 (2)  You can specify a maximum of 3 real device numbers.

 (3)  You can specify a maximum of 3 port names.
```

Refer to *z/VM CP Command and Utility Reference* for full details of all parameters.

SET VSWITCH

The syntax of the SET VSWITCH command is shown in Example 23.

Example 23 SET VSWITCH syntax

```
SET VSWITCH

                                               .-VLAN--ANY--------.
 >>--SET VSWITCH--switchname--.-GRAnt--userid--+------------------+-.--------><
                              |                |       v--------. | |
                              |                '-VLAN----vlanid-+-' |
                              |-REVoke--userid----------------------|
                              |           .--------------.          |
                              |           V          (1) |          |
                              |-PORTname----portname-----+----------|
                              |         .----------.                |
                              |         V      (2) |                |
                              |-RDEV--.---rdev-----+.---------------|
                              |       '-NONE--------'               |
                              |-CONnect-----------------------------|
                              |-DISCONnect--------------------------|
                              |-QUEuestorage--numberM---------------|
                              |-CONTRoller--.-*-------.-------------|
                              |             '-userid1-'             |
                              |-IPTimeout--nnn----------------------|
                              |-NONrouter---------------------------|
                              '-PRIrouter---------------------------'

 Notes:
 (1)  You can specify a maximum of 3 port names.

 (2)  You can specify a maximum of 3 real device numbers.
```

Refer to *z/VM CP Command and Utility Reference* for full details of all parameters.

DIRMAINT NICDEF

Example 24 shows the options available on the DIRMAINT NICDEF command.

Example 24 DIRMAINT NICDEF command

```
NICDEF

>>--DIRMaint--.--------------------.--NICDEF--vdev--------------------------->
              |                 (1)|
              '-Prefix Keywords----'

>--.-DELETE---------------------------------------.----------------------><
   '-TYPE--.-HIPERsockets-.--.--------------------.-'
           '-QDIO---------'  '-| NICDEF Options |-'

NICDEF Options:
   v---------------------------------.
|---.-DEVices--devs-----------------.-+-------------------------------------|
    |-LAN--.-.-ownerid-.--lanname-.-|
    |      | |-*-------|          | |
    |      | '-SYSTEM--'          | |
    |      '-SYSTEM--switchnm-----' |
    |-CHPID--hh---------------------|
    '-MACID--macaddress-------------'

Note:
(1)  For more information on prefix keywords, enter the following command:

       dirm help dirmaint
```

vconfig

Example 25 shows the usage of the **vconfig** command.

Example 25 Usage of the vconfig command

```
lnxsu6:~ # vconfig
Expecting argc to be 3-5, inclusive.  Was: 1

Usage: add             [interface-name] [vlan_id]
       rem             [vlan-name]
       set_flag        [interface-name] [flag-num]       [0 | 1]
       set_egress_map  [vlan-name]      [skb_priority]   [vlan_qos]
       set_ingress_map [vlan-name]      [skb_priority]   [vlan_qos]
       set_name_type   [name-type]

* The [interface-name] is the name of the ethernet card that hosts
  the VLAN you are talking about.
* The vlan_id is the identifier (0-4095) of the VLAN you are operating on.
* skb_priority is the priority in the socket buffer (sk_buff).
* vlan_qos is the 3 bit priority in the VLAN header
* name-type:  VLAN_PLUS_VID (vlan0005), VLAN_PLUS_VID_NO_PAD (vlan5),
              DEV_PLUS_VID (eth0.0005), DEV_PLUS_VID_NO_PAD (eth0.5)
* bind-type:  PER_DEVICE  # Allows vlan 5 on eth0 and eth1 to be unique.
              PER_KERNEL  # Forces vlan 5 to be unique across all devices.
* FLAGS:  1 REORDER_HDR  When this is set, the VLAN device will move the
            ethernet header around to make it look exactly like a real
            ethernet device.  This may help programs such as DHCPd which
            read the raw ethernet packet and make assumptions about the
            location of bytes.  If you don't need it, don't turn it on, because
            there will be at least a small performance degradation.  Default
            is OFF.
```

The author of this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Vic Cross is the Linux for zSeries and S/390® Team Leader at Independent Systems Integrators, IBM's Large Systems Business Partner in Australia. He has more than 15 years of experience in general computing, eight of which have been spent working on S/390 and zSeries. He holds a Bachelor of Computing Science degree from Queensland University of Technology. His areas of expertise include networking and Linux.

He is a co-author of IBM Redbooks™ and Redpapers *Linux on IBM @server zSeries and S/390: ISP/ASP Solutions*, SG24-6299, *Linux on IBM @server zSeries and S/390: Large Scale Linux Deployment*, SG24-6824, *Linux on IBM @server zSeries and S/390: Porting LEAF to Linux on zSeries*, REDP3627, and *Linux on IBM @server zSeries and S/390: Virtual Router Redundancy Protocol on VM Guest LANs*, REDP3657.

Thanks to the following people for their contributions to this project:

Roy Costa and Greg Geiselhart
International Technical Support Organization, Poughkeepsie Center

Angelo Macchiano and Dennis Musselwhite
z/VM Development, IBM Endicott

Simon Williams
IBM Australia

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this Redpaper.

IBM Redbooks

- *Linux on IBM @server zSeries and S/390: Virtual Router Redundancy Protocol on VM Guest LANs*, REDP3657

Other publications

- *z/VM Virtual Machine Operation*, SC24-5955
- *z/VM CP Command and Utility Reference*, SC24-6008

Notices

This document created or updated on September 3, 2003.

Send us your comments in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an Internet note to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks