Redbooks Paper

Simon Williams

Networking Overview for Linux on zSeries

Introduction

As new hardware is introduced and z/VM® virtualization technology evolves, an ever increasing number of networking options are available to the Linux® on zSeries® operating system. Depending on when you implemented Linux on zSeries, the networking infrastructure you chose might now be superseded by a more advanced technology. This IBM® Redpaper provides an overview of the different networking options and provides recommendations for implementing the latest networking technology available to Linux for IBM @server® zSeries.

Note: Networking options described in this redpaper apply to distributions using the Linux 2.4 kernel. Network device driver configuration is fundamentally different in the Linux 2.6 kernel.

Networking options

The networking options available to Linux on zSeries can be broadly split into two categories: physical hardware and virtualization technology. Physical hardware, as the term suggests, covers physical network interfaces, or in the case of HiperSockets™, a networking implementation that requires zSeries hardware. Virtualization technology covers the networking options available to those users who run Linux in a z/VM environment.

The z/VM operating system can use any of the physical networking options. Linux systems running as virtual machines in a z/VM environment have the choice of using any of the physical options, any of the virtualization technology options, or a combination of both, including:

- Physical networking options:
  - Open Systems Adapter-2 and Open Systems Adapter-Express
  - Channel-to-channel adapter
  - Common Link Access to Workstations (CLAW)
  - HiperSockets

- Virtualization technology:
  - Point-to-point connectivity
  - Guest LAN
  - z/VM Virtual Switch (VSWITCH)
  - Layer 2 LAN Switching

Physical networking options

In this section, we look at the type of physical networks available for Linux on zSeries.

Open Systems Adapter-2 (OSA-2)

The Open Systems Adapter-2 (OSA-2) card was designed to provide direct, industry-standard network connectivity for the S/390® server. Figure 1 shows an overview of OSA-2 card connectivity. The OSA-2 card supports ATM, Ethernet, FDDI, and token ring. The two most common types of OSA-2 card are:

- OSA-2 ENTR (Ethernet/token ring)
- OSA-2 FENET (Fast Ethernet)

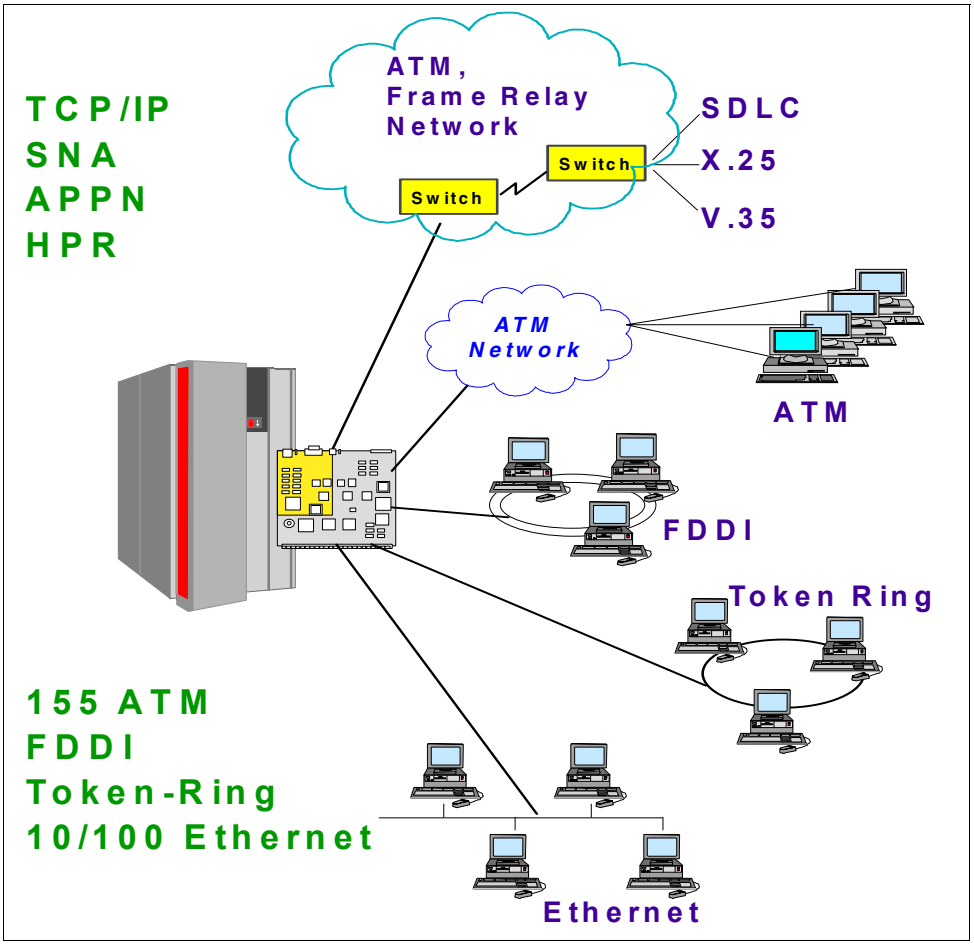

Figure 1 OSA-2 card connectivity overview

OSA-2 ENTR

When the ENTR card is installed on IBM 9672 G5/G6 servers, the physical ports of the card can be configured as one of the following:

- Two 10 Mbps Ethernet ports
- Two 4/16 Mbps token-ring ports
- One 10 Mbps Ethernet port and one 4/16 Mbps token-ring port

Note: When the OSA-2 ENTR card is installed on an IBM @server zSeries 900 server, this card is called the OSA-2 token-ring feature and can only be configured as two 4/16 Mbps token-ring ports.

The z900 is the last server to provide any OSA-2 features. The z800, z890, and z990 do *not* support OSA-2. Instead, there is a choice of OSA-Express features.

Although the ENTR card has two physical ports, a single CHPID is associated with both ports.

OSA-2 FENET

The OSA-2 FENET card supports connections to either a 100 Mbps or 10 Mbps Ethernet LAN. The card has one physical port, with one CHPID associated with that port. The card uses auto-negotiation to set the LAN speed and the duplex mode (either half duplex or full duplex) of the port. If the attached Ethernet switch does not support auto-negotiation, the OSA-2 FENET card enters the LAN at 100 Mbps and half duplex mode.

Note: The OSA-2 FENET card is not supported on the zSeries processors.

The OSA-2 card is classified as an Interconnect Controller. As such, it logically has the same I/O architecture as an IBM 3172, channel-attached routers (for example, Cisco CIP), and the OSA-Express card configured in non-QDIO mode.

As can be seen from Figure 2 on page 4, the OSA-2 card includes the channel layer and control unit function and has a LAN device driver. It uses the Channel Request Handler (CRH) bus to access the application (for example, TCP/IP stack) running on the host.

The CRH bus is a 17 MBps infrastructure designed for ESCON® that provides bandwidth for actual data rates up to 155 Mbps, even though the theoretical ESCON 17 MBps is 170 Mbps. The actual bandwidth of the OSA-2 FENET is, at best, only about 100 Mbps. Obviously, this is insufficient in the modern world of data communications.

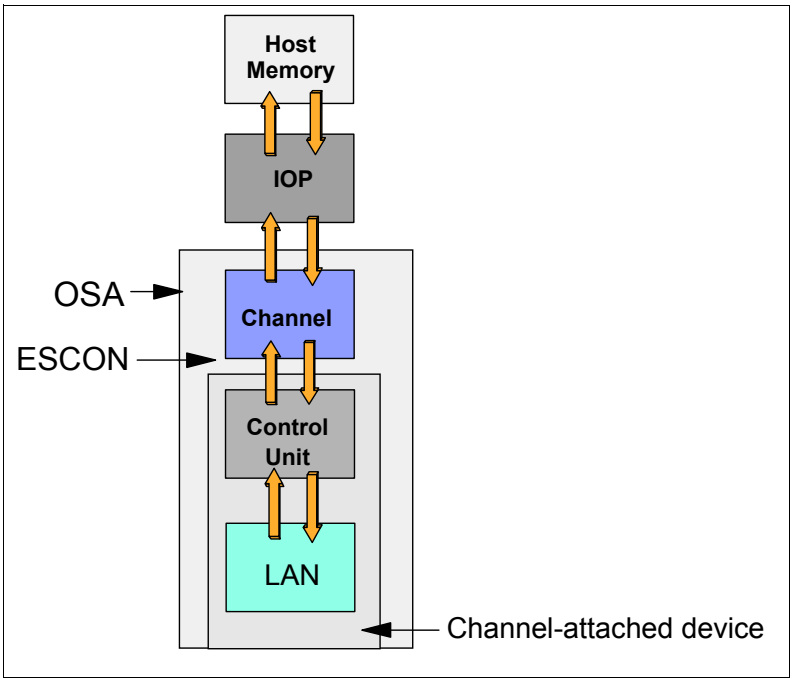

Figure 2 Logical view of OSA-2 card

OSA Address Table (OAT)

The default operating mode of the OSA-2 card is TCP/IP Passthru (non-shared port). In this mode, an OSA-2 port is capable of transferring TCP/IP LAN traffic to and from just one TCP/IP host. You could use this mode if you intend to dedicate an OSA-2 port to a single TCP/IP stack running on an LPAR (for example, Linux or VM TCP/IP).

By exploiting the S/390 Enhanced Multiple Image Facility (EMIF), the OSA-2 card can be shared by multiple LPARs concurrently. Each LPAR (or guest if Linux is running under VM) has a unique IP address, but attaches to the same physical LAN. This operating mode is known as TCP/IP Passthru (shared port) mode. In order for the card to deliver the packets to the correct destination, there must be a map relating IP addresses to an LPAR's device addresses. This map is called the OSA Address Table (OAT). The OAT has a number of rows. Each row associates an IP address with a combination of LPAR, device address, and port.

To configure the OAT for shared mode, the administrator must use Open Systems Adapter/Support Facility (OSA/SF). OSA/SF is a program product that runs on z/OS® or z/VM.

Restriction: If you use OSA-2, be aware that these cards can only support 16 IP addresses per port. This limits the number of guests or LPARs that can share the port. If this limitation affects your deployment of Linux for zSeries, we recommend moving to OSA-Express hardware. Refer to the Washington Systems Center Flash *TCP/IP Stack Limitation on OSA-Express* for detailed information. The full text of that document can be found at:

`http://www.ibm.com/support/techdocs/atsmastr.nsf/PubAllNum/Flash10144`

For a complete description about how to use OSA/SF, refer to *OS/390 V2R7.0 OSA/SF User's Guide*, SC28-1855-06, or *VM/ESA OSA/SF User's Guide*, SC28-1992-03.

Hardware configuration

The IOCP statements in Figure 3 show an OSA-2 ENTR card, CHPID BC. The card is being shared by two LPARs, “ZOS1” and “ZVM1”. A pair of devices are used to represent an OSA-2 network interface. One device is used for read I/O, the other device for write I/O. Using EMIF, devices can be shared across multiple LPARs, in our example, devices 2280 and 2281. However, within a single LPAR, device addresses must be unique, which is why that for a second interface, LPAR “ZVM1” uses devices 2282 and 2283. There is also device 228F, which is defined as an OSAD device (using unit address “FE”). This device is used by the OSA/SF software to communicate with the OSA card.

```
ID MSG1=' IOCP DECK' ,
MSG2=' SYS6.IODFF1 - 04-03-02 15:25'
RESOURCE PARTITION=((ZOS1,2),(ZVM1,8))
CHPID PATH=(BC),SHARED,PARTITION=((ZOS1,ZVM1),(ZOS1,ZVM1)),TYPE=OSA
CNTLUNIT CUNUMBR=2280,PATH=(BC),UNIT=OSA
IODEVICE ADDRESS=(2280,004),UNITADD=00,CUNUMBR=(0220),STADET=Y,UNIT=OSA
IODEVICE ADDRESS=228F,UNITADD=FE,CUNUMBR=(2280),STADET=Y,UNIT=OSAD
```

Figure 3 OSA-2 TCP/IP Passthru mode (shared port): IOCP statements

Figure 4 provides a graphical representation of this configuration.

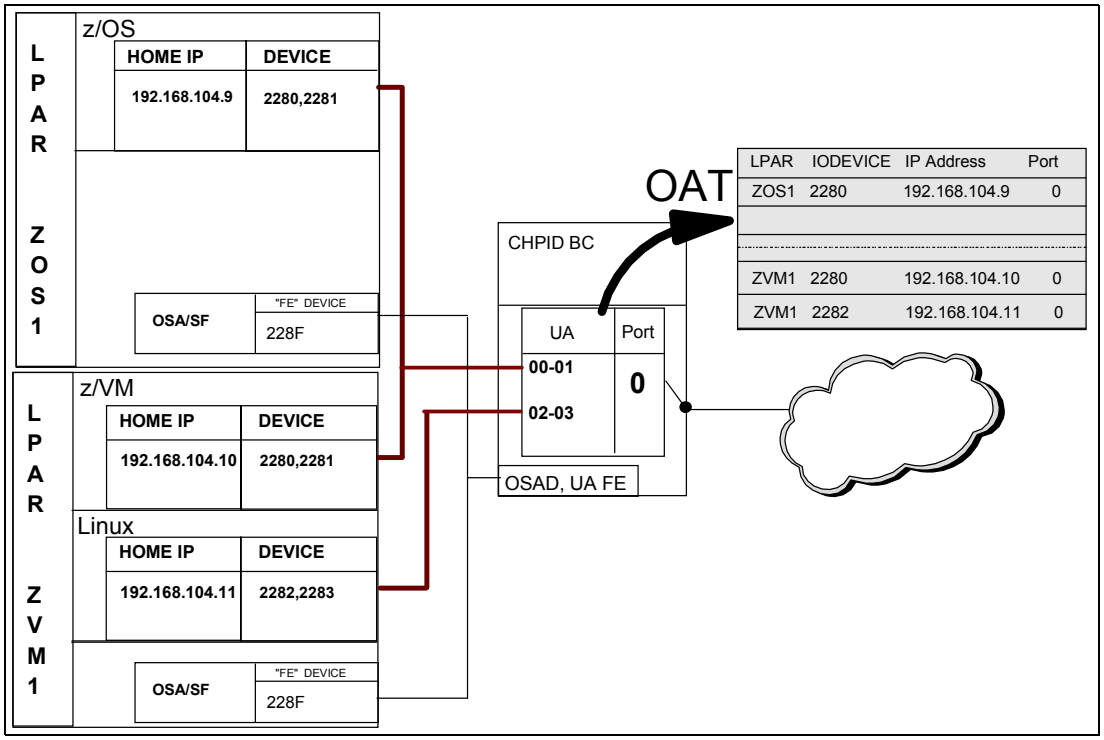

Figure 4 OSA-2 in TCP/IP Passthru shared port mode

An in-depth review of the OSA-2 card is beyond the scope of this Redpaper. Refer to *OSA-2 Implementation Guide (Update)*, SG24-4770.

Using OSA-2 with Linux

Restriction: If a Linux system running in an LPAR is booted very shortly after a power-on-reset, the OSA card might not yet be fully IMLed. When running Linux in an LPAR, you might need to specify an ipldelay=xx kernel boot parameter. We recommend a value between 2m and 5m (between 2 and 5 minutes) for “xx” for the OSA card to initialize fully after IPL. For more details about the ipldelay parameter, refer to the “June 2003 stream” version of *Linux for zSeries and S/390, Device Drivers and Installation Commands, October 7, 2004*, LNUX-1313-04.

This manual can be downloaded from the following URL:

`http://www10.software.ibm.com/developerworks/opensource/linux390/docu/lx24jun03dd04.pdf`

In order to use the OSA-2 card, Linux for zSeries uses the LAN Channel Station (LCS) device driver (lcs.o). Figure 5 shows a summary of the most common network device drivers available to Linux on zSeries. Configure the LCS device driver by passing information to the channel device layer.

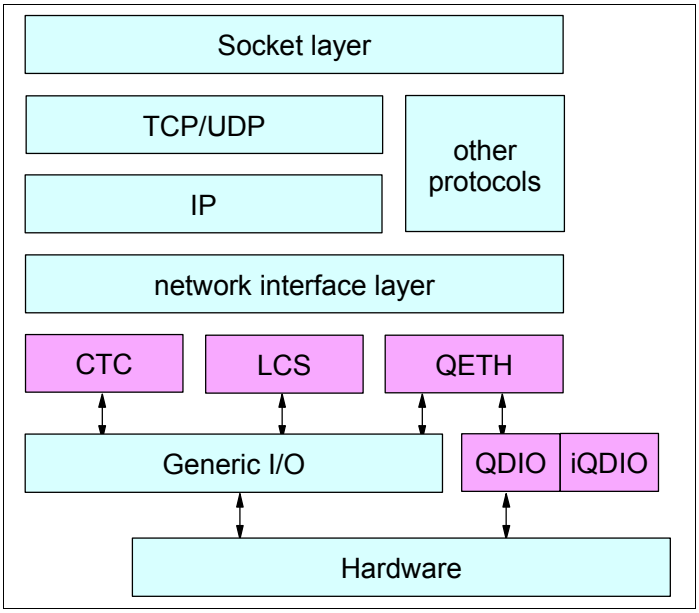

Figure 5 Common network device drivers

Channel device layer

The channel device layer provides a common interface to zSeries and S/390 channel-attached devices. It connects the channel-attached networking devices (CTC/ESCON, LCS, and OSA-Express) to the device drivers. It is activated when a device is attached or detached and when data is written to the /proc/chandev file. At system boot, the channel device layer reads the /etc/chandev.conf file.

Tip: For a detailed review of the channel device layer and device driver syntax, refer to the appropriate level of the Linux on zSeries device drivers manual, as previously mentioned.

There are three ways to pass configuration settings to the channel device layer:

- By redirecting the settings from a shell command line (using echo) to the /proc/chandev file. This file holds the current configuration of the channel device layer. This method is used to dynamically update the channel device layer on a running system.
- By adding the settings to the /etc/chandev.conf file. This file is read at Linux boot time.
- By using the `chandev=` keyword on the Linux kernel boot command line.

An example /etc/chandev.conf definition for an OSA-2 device would look as follows:

```
noauto;lcs-1,0x2280,0x2281,0,1,1,1
```

Table 1 provides a description of the parameters.

Table 1 LCS device driver parameters

Parameter	Description
`noauto`	Stops auto-detection of channel devices.
`lcs-1,0x2280,0x2281,0,1,1,1`	
`lcs-1`	The device interface number. A value of "-1" indicates that the next available device number will be automatically allocated. For example, if we already had the lcs0 and lcs1 devices defined to the channel device layer, the next device to be defined would be lcs2.
`0x2280`	The read device address.
`0x2281`	The write device address.
`0,`	The number of kilobytes to be allocated for read and write buffers. The default 0 means that the driver itself determines this number.
`1,`	The relative adapter number of LCS. For an OSA-2 device this could be either port 0 or port 1.
`1,`	IP checksumming is enabled.
`1`	Network statistics collection is enabled.

Recommendations

If you are operating in a hardware environment where you only have OSA-2 technology, directly connecting your Linux LPARs or Linux guests to the OSA-2 card or cards will be the most efficient means of accessing the network.

The alternative approach is to provide connectivity to the Linux guests through z/OS, z/VM, or a Linux guest acting as an intermediate router. That router machine "owns" the OSA-2 interface and the "back-end" Linux systems are connected to that router through a virtualization technology (virtual CTC, IUCV, or Guest LAN). Although this router can provide added functionality such as packet filtering and VPN support, it also adds latency and extra CPU overheard to your environment.

One point to note, however, is that OSA-2 cards can only support 16 IP addresses per port. This limits the number of guests or LPARs that can share the port.

OSA-2 cards, introduced in 1995, use the Interconnect Controller architecture and thus are not as efficient as OSA-Express cards running in QDIO mode. The OSA-2 Fast Ethernet card, under the best conditions, has a maximum bandwidth of 100 Mbps. The OSA-Express Gigabit Ethernet and 1000BASE-T cards have, by comparison, a maximum bandwidth of 1000 Mbps. With that in mind, our recommendation is that you move to OSA-Express Gigabit Ethernet or 1000BASE-T.

Open Systems Adapter-Express (OSA-Express)

The Open Systems Adapter-Express (OSA-Express) Gigabit Ethernet (GbE), 1000BASE-T Ethernet, Fast Ethernet (FENET), token-ring, and Asynchronous Transfer Mode (ATM) cards are the next generation cards that supersede the OSA-2 family of cards. OSA-Express cards provide significant enhancements over OSA-2 in function, connectivity, bandwidth, network availability, reliability, and recovery. OSA-Express cards are available for all zSeries processors and S/390 G5 and G6 processors. See Figure 6 on page 9 for a diagram of OSA-Express connectivity options.

Each OSA-Express card has one port on G5 and G6 servers and two ports on zSeries servers. They can be attached directly to a LAN or ATM network. These cards are recognized by the hardware I/O configuration as one of the following channel types:

- OSD (Queued Direct I/O)
- OSE (Non-Queued Direct I/O)

Note: OSA-2 cards have two ports, but have only a single CHPID associated with both ports. OSA-Express cards have two ports, but have a unique CHPID associated with each port.

OSA-Express cards on the zSeries 990 processor operating in QDIO mode support up to 160 stacks and 480 devices per port. Refer to the Washington Systems Center Flash *TCP/IP Stack Limitation on OSA-Express* for detailed information, available at:

http://www.ibm.com/support/techdocs/atsmastr.nsf/PubAllNum/Flash10144

QDIO mode

Queued Direct I/O (QDIO) is a highly efficient data transfer mechanism. It reduces system overhead and improves throughput by using system memory queues and a signaling protocol to directly exchange data between the OSA-Express microprocessor and TCP/IP stack.

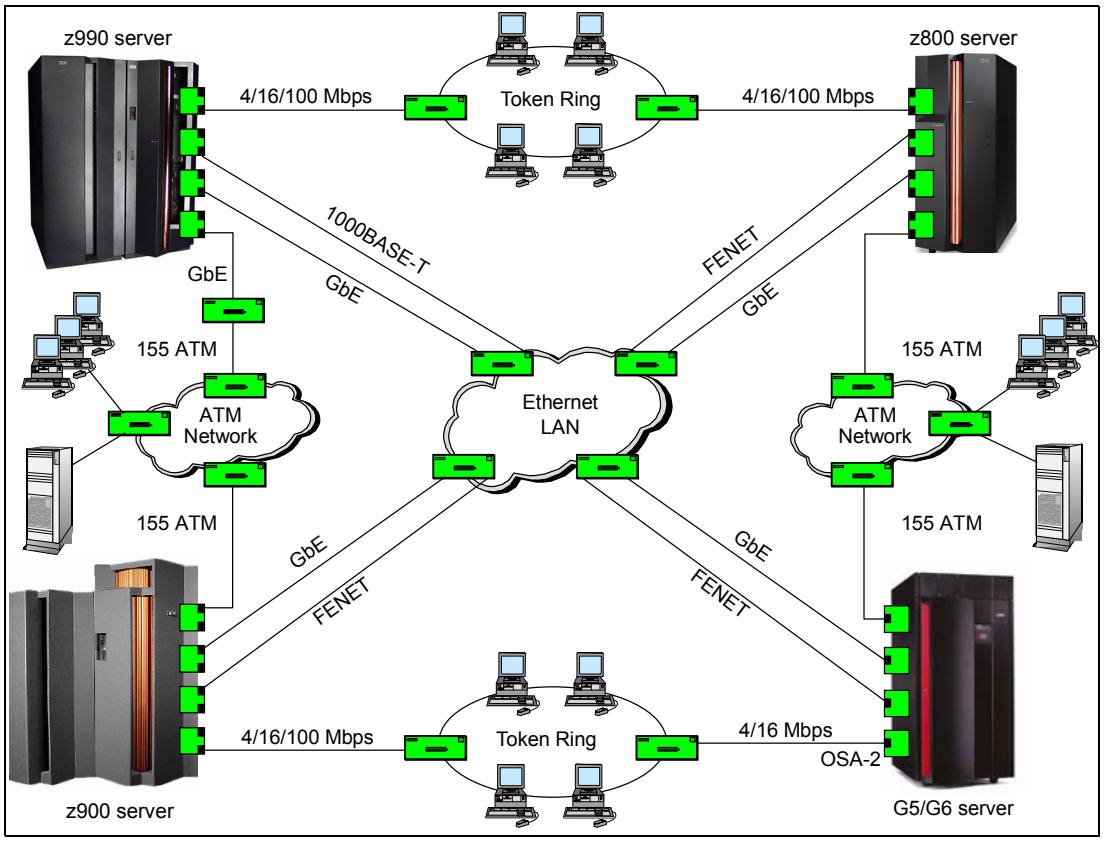

Figure 6 OSA-Express connectivity

QDIO versus non-QDIO

Figure 7 on page 10 illustrates the much shorter I/O path length of the QDIO-enabled card compared with the non-QDIO card (which has the same I/O path length as the OSA-2 cards). Consequently, when running in QDIO mode, I/O interrupts and I/O path lengths are minimized. When running in QDIO mode, measurements have shown that there is a significant improvement in performance versus non-QDIO mode, in particular, a reduction of System Assist Processor (SAP) utilization and improved response time.

Benefits of running in QDIO mode include:

- Dynamic OSA Address Table (OAT) update
- LPAR-to-LPAR communication
- Internet Protocol (IP) Assist functions
- Checksum offload

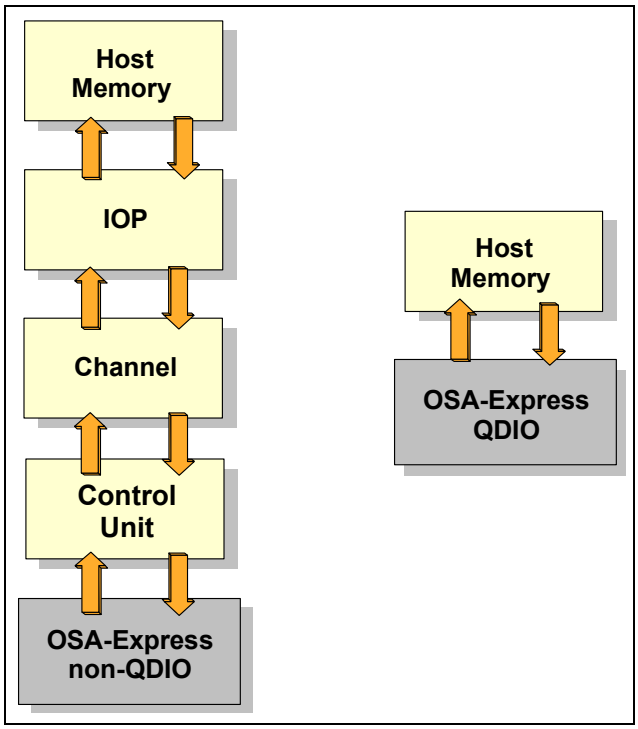

Figure 7 QDIO and non-QDIO data paths

Dynamic OSA Address Table (OAT) update

The TCP/IP stack of each operating system that shares a port on an OSA-Express card in QDIO mode dynamically registers all its IP addresses with the card. Whenever IP addresses are deleted from or added to a network stack, the device drivers download the resulting IP address list changes to the OSA-Express card.

For OSA-Express cards shared by multiple systems, this removes the requirement to manually enter the information into the OAT using OSA/SF. A user might still have a requirement to use OSA/SF however, for example, if you need to enter SNA definitions into the card when the card is running in non-QDIO mode.

LPAR-to-LPAR communication

Using EMIF, a port on the OSA-Express card can be shared across multiple LPARs, as depicted in Figure 8 on page 11. Also, access to a port on the card can be shared concurrently among multiple TCP/IP stacks within the same LPAR.

When port sharing, the OSA-Express card running in QDIO mode has the ability to send and receive IP traffic between LPARs without sending the IP packets over the network.

For outbound packets, OSA-Express uses the next-hop address provided by the TCP/IP stack to determine where to send the packet. If this next-hop address had been registered by another TCP/IP stack sharing this OSA-Express, the packet will be delivered directly to that TCP/IP stack, and not sent out over the LAN. This makes possible the routing of IP packets within the same host system.

Note: LPAR-to-LPAR communication also applies to OSA-Express FENET when the mode is non-QDIO.

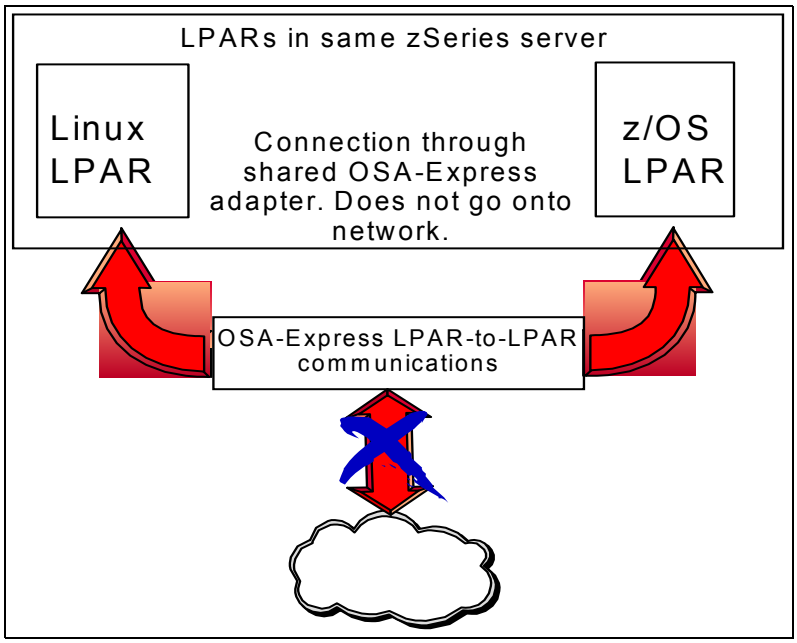

Figure 8 LPAR-to-LPAR communication

Internet Protocol (IP) Assist functions

The OSA-Express QDIO microcode assists in IP processing and offloads the TCP/IP stack functions for the following:

- Multicast support
- Broadcast filtering
- Building MAC and LLC headers
- ARP processing

Offloading the processing of these functions means that CP cycles are freed up to do other work. In a single guest, the effect might not be significant, but in a z/VM LPAR with Linux guests generating a moderate-to-high volume of network traffic, there will be an overall saving.

Checksum offload for Linux and z/OS

Checksum processing calculates the TCP/UDP and IP header checksums to verify the integrity of data packets. This function is usually performed by a host system's TCP/IP stack. OSA-Express cards on the z990 and z890 processors have the ability to perform checksum processing on behalf of the upstream TCP/IP stack using a function called checksum offload. This function is only available for IPv4 packets.

By moving the checksum calculations to an OSA-Express Gigabit or 1000BASE-T Ethernet card, host CPU cycles are reduced. This support is available with z/OS V1R5 and later and Linux for zSeries.

Note: In order to use checksum offload with Linux for zSeries, you must use the qeth module's parameter hw_checksumming. Linux for zSeries supports inbound checksum offload (inbound packets) only. Refer to *Linux for zSeries and S/390, Device Drivers and Installation Commands, October 7, 2004*, LNUX-1313-04, for additional information about the hw_checksumming device driver parameter.

Non-QDIO mode

When running in non-QDIO mode, a port on the OSA-Express card is defined as channel type OSE.

Note: The zSeries OSA-Express cards have two ports. Each port has an associated CHPID. It is possible to configure one CHPID as type OSD (QDIO) and one CHPID as OSE (non-QDIO), or both CHPIDs as OSD or OSE.

In non-QDIO mode, the data follows the same logical I/O path as an OSA-2 card. Linux uses the LCS device driver to communicate with the device when it is running in this mode. The non-QDIO mode requires the use of OSA/SF for customization of the OSA-Express if you want to share the card across multiple LPARs or Linux guests.

The OSA-Express 1000BASE-T, FENET, and token-ring cards support both non-QDIO and QDIO modes. The OSA-Express Gigabit Ethernet card only supports QDIO mode.

Unless you have a specific requirement (such as supporting SNA traffic), we recommend that you always run the OSA-Express card in QDIO mode.

Address Resolution Protocol and OSA-Express

Address Resolution Protocol (ARP) is a networking protocol used to resolve IP addresses to physical hardware addresses. These hardware addresses are known as Media Access Control (MAC) addresses.

When an application running on machine X wants to send a datagram to machine Y, it typically uses machine Y's IP address as the address that it uses to try and reach the destination. However, the device driver controlling the Network Interface Card does not understand IP addresses and wants to send the datagram using a MAC address as the destination address.

The ARP protocol attempts to resolve the IP address into a MAC address. It does this by referencing a lookup table (called an ARP cache). If the address is not found in the ARP cache, an ARP request is broadcast over the network. If one of the machines that receives the broadcast recognizes its own IP address, it will answer the requesting machine with an ARP reply message. This reply will include the MAC address of that host. This information is then stored in the requesting system's ARP cache. Any subsequent datagrams to this destination IP address will be translated to a MAC address by referring to the ARP cache.

In the majority of computer systems, a network card is "owned" by a single TCP/IP stack; therefore, there is a one-to-one relationship between the IP address and MAC address. Figure 9 on page 13 illustrates MAC address and IP address processing in a distributed environment.

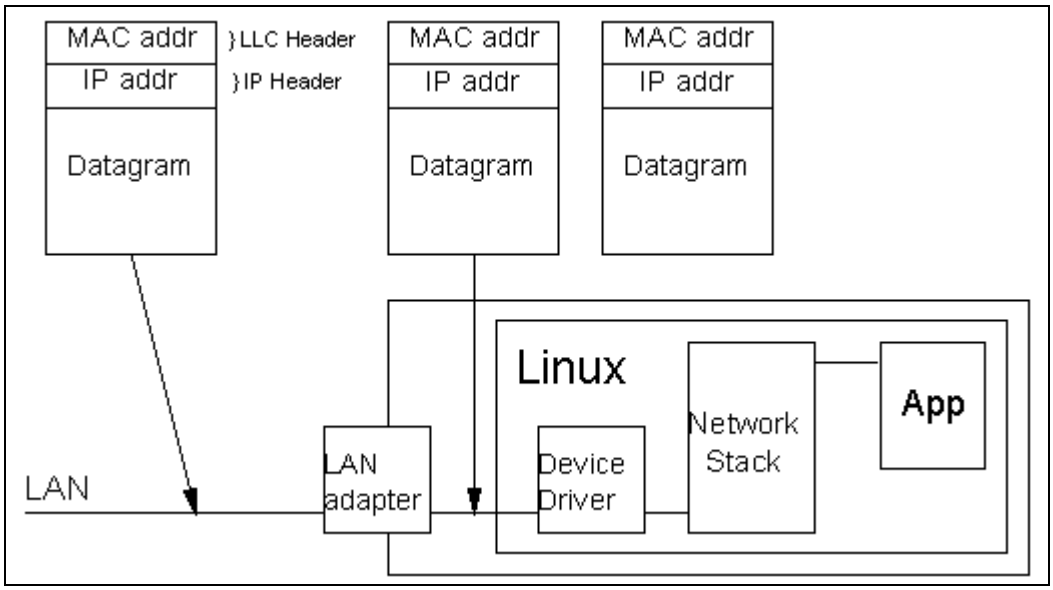

Figure 9 Packet processing in non-mainframe environments

OSA cards can be shared across multiple LPARs or guests. Each TCP/IP stack running in these systems has a unique IP address that is dynamically registered with the OSA-Express card.

All of the IP addresses registered in the OSA-Express card are associated with the same MAC address.[1] The OSA-Express card will respond to ARP requests from other machines in the network for any IP address that is registered in the card.

The OSA-Express card removes the Logical Link Control (LLC) header, which includes the MAC address from incoming IPv4 packets, and uses the registered IP address to forward packets to the recipient TCP/IP stack. This is how the card delivers IPv4 packets to the correct Linux image. Apart from broadcast packets, a Linux image can only receive packets for IP addresses it has configured in the stack and registered with the OSA-Express card.[2]

As the OSA-Express QDIO microcode builds LLC headers for outgoing IPv4 packets and removes them from incoming IPv4 packets, the operating system's network stacks only send and receive IPv4 packets without LLC headers. Figure 10 on page 14 illustrates MAC address and IP address processing by OSA-Express.

[1] See "Layer 2 LAN Switching" on page 53 for details about the new Layer 2 Switching technology that allows Linux guests running under z/VM to have unique MAC addresses.
[2] See "Primary and secondary router function" on page 14 for the only exception to this statement.

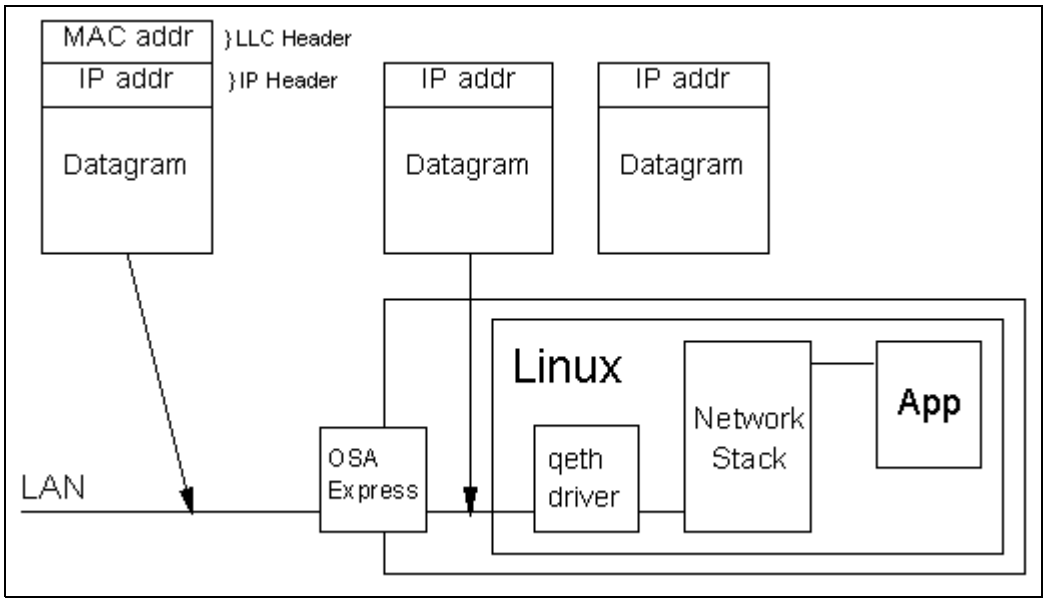

Figure 10 Packet processing by OSA-Express

Letting the OSA-Express hardware handle the LLC header allows multiple operating systems to share an OSA-Express adapter card. Usually, LLC processing by the OSA-Express card also yields better performance than letting the Linux images that share the card handle the LLC header themselves. For IPv6, the OSA-Express card in QDIO mode passes complete packets to the Linux image, and the driver lets the network stack compose packets with an LLC header.

Primary and secondary router function

A port on an OSA-Express card can be configured to forward unknown IP addresses to a particular TCP/IP stack for routing. For example, a Linux system could act as a router to provide a means of connecting an external LAN segment to systems running within a zSeries machine on a different subnet.

In order for the OSA-Express port to forward datagrams to a specific TCP/IP stack, the Linux system must have loaded the OSA-Express device driver with the primary_router parameter. If the OSA-Express port is being shared by multiple systems, only one of those systems can act as the primary router. Refer to *Linux for zSeries and S/390, Device Drivers and Installation Commands, October 7, 2004*, LNUX-1313-04, for detailed information about the primary_router parameter.

Hardware configuration

The IOCP statements in Figure 11 show an OSA-Express card, CHPID 01. The card is being shared by two LPARs.

```
ID MSG1=' IOCP DECK' ,
MSG2=' SYS6.IODFF1 - 04-03-02 15:25'
RESOURCE PARTITION=((SC47,2),(SC69,8))
CHPID PATH=(01),SHARED,PARTITION=((SC47,SC69),(SC47,SC69)),TYPE=OSD
CNTLUNIT CUNUMBR=C100,PATH=(01),UNIT=OSA
IODEVICE ADDRESS=(C100,015),CUNUMBR=(C100),UNIT=OSA
IODEVICE ADDRESS=C10F,UNITADD=FE,CUNUMBR=(C100),UNIT=OSAD
```

Figure 11 OSA-Express IOCP statements

Figure 12 provides a graphical representation of this configuration.

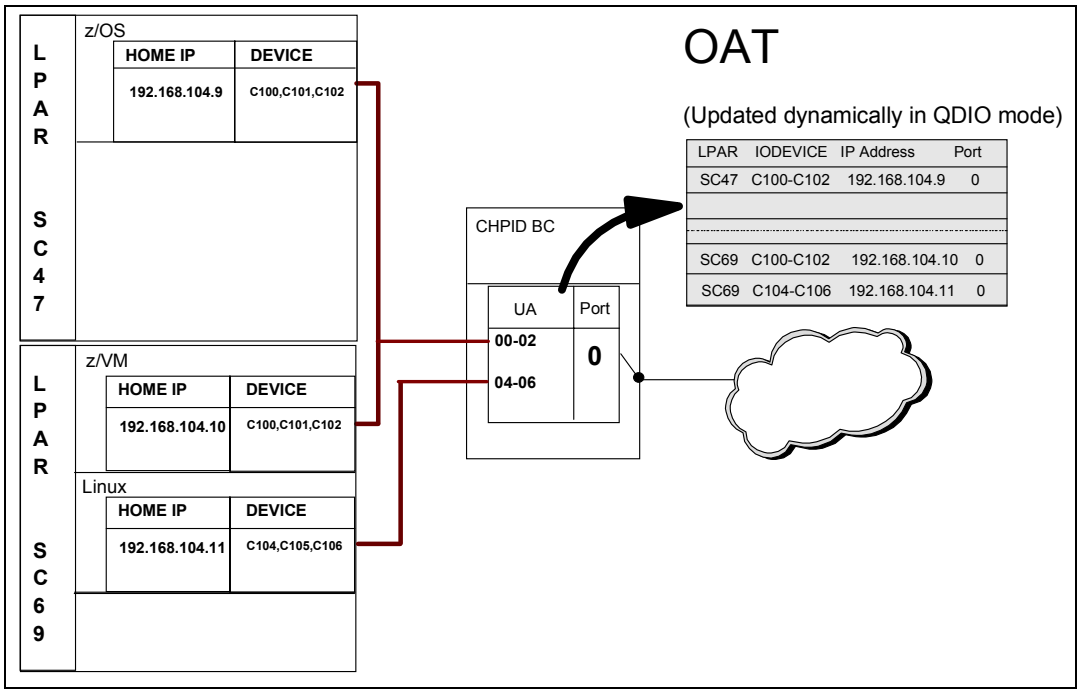

Figure 12 OSA-2 in TCP/IP Passthru shared port mode

Using EMIF, devices can be shared across multiple LPARS (devices C100-C102). However, within a single LPAR, device addresses must be unique. That is why the Linux system on LPAR SC69 uses devices C104-C106. An in-depth review of the OSA-Express card is beyond the scope of this Redpaper. Refer to *OSA-Express Implementation Guide*, SG24-5948.

Important: OSA-2 and non-QDIO mode OSA-Express cards require two devices (read/write) to form a network interface. OSA-Express cards running in QDIO mode require three devices (read/write/data) to form a network interface. At the time of writing this paper, there was a limit of 480 device addresses that could be configured on a z990 OSA-Express CHPID. When you define the CHPID as shared, this limit applies to the total number of devices defined across all LPARs in the candidate list for that CHPID. You must carefully plan which LPARs are defined in the candidate list; otherwise, you might find that a specific LPAR (perhaps running multiple Linux systems under z/VM) does not have enough device addresses available.

z/VM considerations

Multiple virtual machines can share the same physical OSA card. All guests can be defined with the same three *virtual* device addresses for the OSA interface. However, each guest must use a unique set of three *real* device addresses.

The syntax of the DEDICATE statement is:

```
DEDICATE virtual_address real_address
```

For example, you might choose C200-C202 as the *virtual* addresses for all of your guests. Each guest must, however, have unique *real* addresses. So in the first guest, you might use addresses C200-C202, in the second guest, C203-C205, and so on.

Important: The first device address is the OSA read device. It must be an even-numbered device. The second device is the OSA write device, and its address must be one greater than the read device. Using the examples above, you would load the OSA-Express device driver (see next section for more details) with the following parameters for the C200-C202 devices:

```
qeth-1,0xc200,0xc201,0xc202,0,0
```

The C203-C205 devices would use the following parameters:

```
qeth-1,0xc204,0xc205,0xc203,0,0
```

Using this convention, we use every device address and thus do not waste addresses by skipping to the next even-numbered address to start the next device set. For example, C200-C202 followed by C204-C206 would mean we "waste" C203, and therefore, we are not able to fully use the OSA-Express device.

Using OSA-Express with Linux

As previously discussed, a port on an OSA-Express card can run in one of two modes, QDIO or non-QDIO.[3] We will review the parameters for running the card in QDIO mode. In non-QDIO mode, the OSA-Express card acts like as an Interconnect Controller, and Linux for zSeries uses the LAN Channel Station (LCS) device driver (lcs.o) to control the card.

In QDIO mode, the following two modules are required in order for Linux to use the OSA-Express card.

- qdio controls the interface between the processor and the OSA-Express CHPID.
- qeth controls the OSA-Express card.

Two other modules, ipv6 and 8021q, are typically loaded automatically to provide support for IPv6 and Virtual LANs. For our purposes, we only need to configure the qeth module. An example /etc/chandev.conf configuration for an OSA-Express port running in QDIO mode follows:

```
noauto;qeth-1,0xc300,0xc301,0xc302,0,0;add_parms,0x10,0xc300,0xc302,portname:OSACHP03
```

Table 2 on page 4 describes the device parameters.

Table 2 qeth device driver parameters

Parameter	Description
`noauto`	Stops auto-detection of channel devices.
`qeth-1,0xc300,0xc301,0xc302,0,0`	
`qeth-1`	The device interface number. A value of "-1" indicates that the next available device number will be automatically allocated. For example, if we already had qeth0 and qeth1 devices defined to the channel device layer, the next device to be defined would be qeth2.
`0xc300`	The read subchannel address.
`0xc301`	The write subchannel address.
`0xc302`	The data subchannel address.

[3] The OSA-Express 1000BASE-T, FENET, and token-ring cards support both non-QDIO and QDIO modes. The OSA-Express Gigabit Ethernet card only supports QDIO mode.

Parameter	Description
0,	The number of kilobytes to be allocated for read and write buffers. 0 specifies the default value (8192 KB in QDIO mode).
0	The relative port number of the CHPID. OSA-Express devices use only port 0.
add_parms,0x10,0xc300,0xc301,portname:OSACHP03	
add_parms	Used to pass additional parameters to the driver.
0x10	Identifies the device as an OSA-Express CHPID in QDIO mode.
0xc300,0xc302	The desired device address range.
portname:OSACHP03	Identifies the port for sharing by other operating system images. The port name can be 1 to 8 characters long and must be uppercase. All operating systems sharing the port must use the same port name.[a]

a. This parameter is required only for S/390 G5 and G6 processors and for zSeries 800 and 900 machines that are below a particular maintenance level. For z890, z990, and later mainframes, you are advised to omit it.

OSA port names

For z800 and z900 processors, PORTNAME is no longer required if you have driver 3G, EC stream J11204 MCL032 (OSA Level 3.33) installed (with the appropriate levels of z/VM and Linux). Refer to the Washington Systems Center Flash *OSA-Express MCL Enhancements - October 2003* for detailed information. The full text of the WSC Flash can be found at:

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/FLASH10250

OSA-Express layer 2 feature

zSeries Layer 2 support provides Layer 2 (Ethernet) transport for Linux through the virtualization technology in both z/VM and the OSA-Express Adapter. OSA-Express provides the Layer 2 trunk support required by the z/VM Virtual Switch and Linux for external LAN connectivity along with MAC and VLAN filtering. In Layer 2 mode, the OSA-Express:

- Uses the MAC destination address to send and receive Ethernet frames.
- Transports Ethernet frames (not IP datagrams) to and from the VSWITCH/Linux connections and the physical network.
- Does not ARP offload; ARP processing performed by Linux.
- Supports MAC level unicast, multicast, and broadcast.

The z/VM Virtual Switch, while providing a Layer 2 virtual networking fabric, also provides MAC address generation and assignment along with IEEE VLAN support from both an authorization and deployment of virtual LAN segments. In concert with z/VM and OSA-Express, the Linux qeth device driver for the OSA-Express feature was enhanced to exploit this new Layer 2 networking capability in support of IP and non-IP based applications.

Recommendations

OSA-Express cards provide significant functional and performance improvements over the earlier OSA-2 family of cards. For any applications that have large bandwidth requirements, we recommend that you use OSA-Express.

Whenever you have a requirement to directly connect your Linux systems or z/VM TCP/IP stack to an external network, we strongly recommend that you move to OSA-Express Gigabit or 1000BASE-T if you have not already done so.

Channel-to-channel adapter

Channel-to-channel (CTC) is a point-to-point connection, using real hardware channels. It is used to interconnect different physical servers, logical partitions, or both. Because all zSeries operating systems use the same link protocol, it is possible to connect a Linux server not only to another Linux, but also to a z/VM or z/OS TCP/IP stack. CTC support exists for a number of channel technologies including ESCON and FICON® channels.

ESCON CTC connectivity

To connect two systems using ESCON, you must define two channels. On one side, the channel is defined as CHPID type CTC, on the other side, as CHPID type CNC. The ESCON CTC connection can either be point-to-point or switched point-to-point (that is, it can be connected to an ESCON director).

EMIF allows LPARs to share channel paths, and so optionally, they can share any control units and associated I/O devices configured to these shared channels. Sharing channel paths means that you can reduce the number of physical connections between processor complexes. Both CTC and CNC channels can be defined as shared channels; see Figure 13.

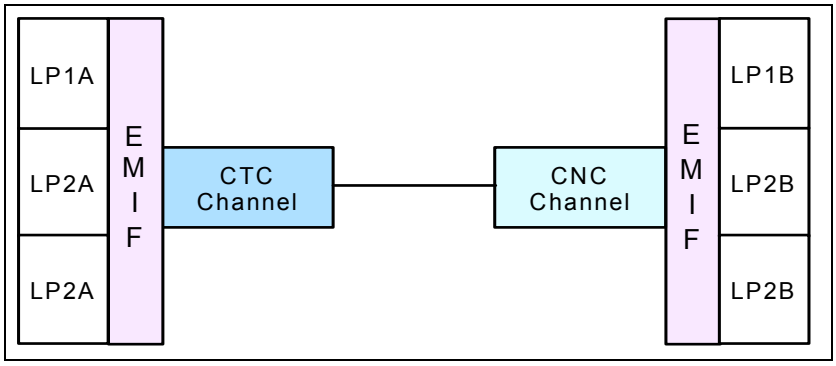

Figure 13 CTC connection with shared channels

FICON CTC connectivity

Channel-to-channel communication in a FICON environment is provided between two FICON (FC) channel FCTC control units.

There are several differences between the ESCON and FICON CTC implementations, as shown in Table 3.

Table 3 ESCON and FICON CTC differences

Characteristic	ESCON	FICON
Number of required channels	At least 2	1 or 2
Channel dedicated to CTC function	Yes	No
Number of unit addresses supported	Up to 512	Up to 16384
Data transfer bandwidth	12-17 MBps	Up to 2 Gbps
Number of concurrent I/O operations	1	Up to 32
Data transfer mode	Half duplex	Full duplex

The details of these differences are as follows:

- ESCON CTC connectivity is provided by a pair of ESCON channels, one defined as CTC and the other defined as CNC. At least two ESCON channels are required.

  FICON CTC connectivity can be implemented using one or two FICON (FC) native channels.

- An ESCON channel defined as CTC can only support the CTC function. Only a control unit (type SCTC) can be defined on an ESCON CTC channel.

  The FICON native (FC) channel supporting the FCTC control unit can communicate with an FCTC control unit on another machine, and simultaneously, the same FICON (FC) channel can also support operations to other I/O control unit types such as DASD and tape.

- An ESCON CTC channel supports a maximum of 512 unit addresses (devices).

  A FICON native (FC) channel supports a maximum of 16,384 unit addresses (devices).

- An ESCON channel has a data transfer bandwidth of 12-17 MB, significantly less than the FICON or FICON Express channels.

- An ESCON channel supports only one actively communicating I/O operation at a time, while the FICON channel supports up to 32 concurrent I/O operations.

- An ESCON channel operates in half duplex mode, transferring data only in one direction at a time. A FICON channel operates in full duplex mode, sending and receiving concurrently.

CTC communication across multiple LPARs using one FICON channel

A single FICON channel connected to a FICON Director can provide the FICON CTC communications between LPARs on a single processor, as well as images on other processors. It can also be used to communicate to other I/O control units. Figure 14 on page 20 shows a sample configuration.

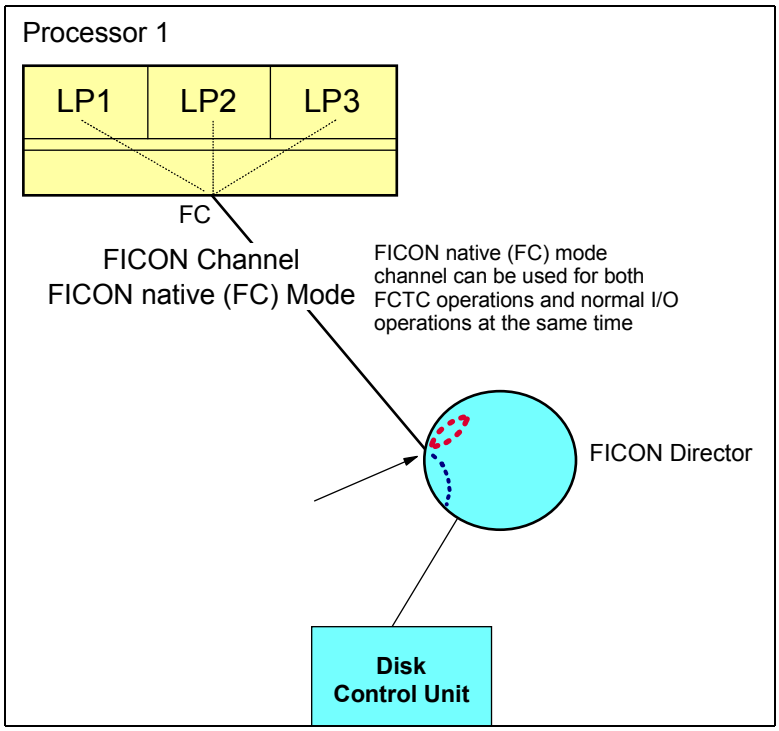

Figure 14 Single FICON channel on one processor

For detailed information about FICON CTC implementation, refer to the Redpaper *FICON CTC Implementation*, REDP-0158.

Each end of the connection requires two devices, a device for read I/O and a device for write I/O. The read device on one side needs to be connected to the write device on the other side and vice versa.

Recommendations

We do not recommend the use of ESCON or FICON CTCs as networking connectivity options for your Linux on zSeries systems. For inter-LPAR communications, we recommend that you use HiperSockets or OSA-Express. For communications inside a single z/VM LPAR, we recommend VSWITCH. Although CTC bandwidth is good (particularly FICON Express), CTC connectivity is less fault tolerant than other solutions. Often, if one side of the link has a problem, one or even both of the systems have to be re-IPLed in order to restart the CTC link. For communications between the zSeries machine and other systems in the network, we recommend that you use OSA-Express Gigabit Ethernet or OSA-Express 1000BASE-T.

Common Link Access to Workstation (CLAW)

Common Link Access to Workstation (CLAW) is a point-to-point protocol. A CLAW device is an ESCON channel-attached device that supports CLAW protocol. These devices can be used to connect your Linux for zSeries system to another system, for example, an RS/6000 or a Cisco Channel Interface Processor (CIP) card.

Tip: For a detailed review of the CLAW device driver syntax, refer to the appropriate level of the device drivers manual. The level of the manual used in this Redpaper is *Linux for zSeries and S/390, Device Drivers and Installation Commands, October 7, 2004*, LNUX-1313-04, available at:

http://www10.software.ibm.com/developerworks/opensource/linux390/docu/lx24jun03dd04.pdf

Recommendations

CLAW devices are “old technology” and are not as efficient or reliable as some other solutions. Instead, for communications between Linux and other systems in the network, we recommend that you use OSA-Express Gigabit or 1000BASE-T.

HiperSockets

HiperSockets provides very fast TCP/IP communications between servers running in different logical partitions (LPARs) on a zSeries machine. The z890 and z990 processors support up to 16 HiperSocket “internal LANs.” The z800 and z900 processors support up to four HiperSockets. Each HiperSocket is defined as a CHPID of type “IQD.”

To communicate between servers running in the same zSeries Central Electronics Complex (CEC), HiperSockets sets up I/O queues in the zSeries processor’s memory. The packets are then transferred at memory speeds between the servers, thereby totally eliminating the I/O subsystem overhead and any external network latency.

HiperSockets implementation is based on the OSA-Express Queued Direct Input/Output (QDIO) protocol; therefore, HiperSockets is called internal QDIO (iQDIO). HiperSockets is implemented in microcode that emulates the Logical Link Control (LLC) layer of an OSA-Express QDIO interface.

So although HiperSockets is a type of virtualization technology, it relies on zSeries microcode to run, and therefore for the purpose of this paper, we categorize it as a physical networking option.

Typically, before a packet can be transported on an external LAN, a LAN frame has to be built, and the MAC address of the destination host or router on that LAN has to be inserted into the frame. HiperSockets does not use LAN frames, destination hosts, or routers. TCP/IP stacks are addressed by inbound data queue addresses instead of MAC addresses. The zSeries server microcode maintains a lookup table of IP addresses for each HiperSocket. This table represents an internal LAN. At the time a TCP/IP stack starts a HiperSockets device, the device is registered in the IP address lookup table with its IP address and its input and output data queue pointers. If a TCP/IP device is stopped, the entry for this device is deleted from the IP address lookup table.

HiperSockets copies data synchronously from the output queue of the sending TCP/IP device to the input queue of the receiving TCP/IP device by using the memory bus to copy the data through an I/O instruction. The controlling operating system that performs I/O processing is identical to OSA-Express in QDIO mode. The data transfer time is similar to a cross-address space memory move, with hardware latency close to zero.

HiperSockets operations are executed on the processor where the I/O request is initiated by the operating system. HiperSockets starts write operations; the completion of a data move is indicated by the sending side to the receiving side with a Signal Adapter (SIGA) instruction. Optionally, the receiving side can use dispatcher polling instead of handling SIGA interrupts. The I/O processing is performed without using the System Assist Processor (SAP). This new

implementation is also called *thin interrupt*. HiperSockets does not contend with other system I/O activity and it does not use CPU cache resources; therefore, it has no association with other activity in the server.

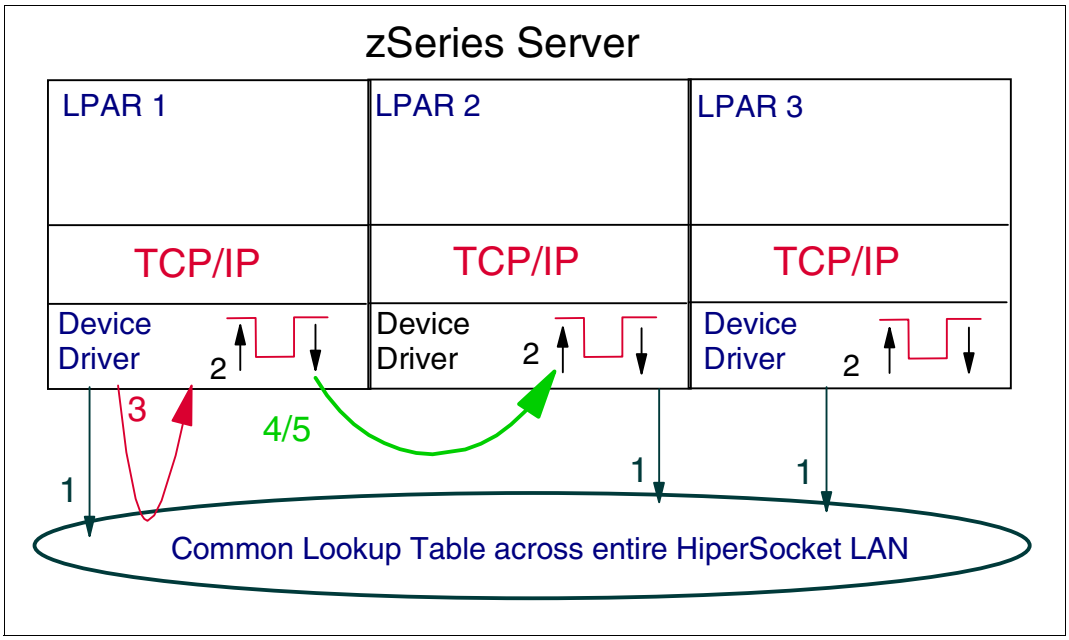

Figure 15 HiperSockets basic operation

The HiperSockets operational flow (represented in Figure 15) consists of five steps:

1. Each TCP/IP stack (image) registers its IP addresses into HiperSockets' server-wide Common Address Lookup table. There is one lookup table for each HiperSockets LAN.
2. The address of the TCP/IP stack's receive buffers are appended to the HiperSockets queues.
3. When data is being transferred, the send operation of HiperSockets performs a table lookup for the addresses of the sending and receiving TCP/IP stacks and their associated send and receive buffers.
4. The sending processor copies the data from its send buffers into the target processor's receive buffers (zSeries server memory).
5. The sending processor optionally delivers an interrupt to the target TCP/IP stack. This optional interrupt uses the "thin interrupt" support function of the zSeries server, which means the receiving host will "look ahead," detecting and processing inbound data. This technique reduces the frequency of real I/O or external interrupts.

For a detailed review of HiperSockets, refer to the Redbook *zSeries HiperSockets*, SG24-6816.

Hardware configuration

The IOCP statements in Figure 16 on page 23 show two HiperSocket LANs. The HiperSocket LANs are being shared by multiple LPARs.

```
ID    MSG1='IODFA2',MSG2='SYS1.IODFA2 - 2004-08-04 03:26',    *
               SYSTEM=(2066,1),                                  *
               TOK=('ZAPHOD',000000011C8A2066032619730104217F00000000,0*
               0000000,'04-08-04','03:26:19','SYS1','IODFA2')
          RESOURCE PARTITION=((CF0A,E),(CF0B,F),(CF01,A),(CF02,B),(LINUX*
               1,9),(VM1,5),(VM2,6),(VM3,7),(VM4,8),(ZOSL,C),(ZOSS,D),(*
               ZOS1,1),(ZOS2,2),(ZOS3,3),(ZOS4,4))
           CHPID PATH=(FA),SHARED,                                *
                 PARTITION=((LINUX1,VM1,ZOS1,ZOS2,ZOS3),(LINUX1,VM1,VM2,V*
                 M3,VM4,ZOSL,ZOSS,ZOS1,ZOS2,ZOS3,ZOS4)),TYPE=IQD
           CHPID PATH=(FB),SHARED,                                *
                 PARTITION=((LINUX1,VM1,ZOS1,ZOS2,ZOS3),(LINUX1,VM1,VM2,V*
                 M3,VM4,ZOSL,ZOSS,ZOS1,ZOS2,ZOS3,ZOS4)),TYPE=IQD
           CNTLUNIT CUNUMBR=FA00,PATH=(FA),UNIT=IQD
           CNTLUNIT CUNUMBR=FB00,PATH=(FB),UNIT=IQD
           IODEVICE ADDRESS=(FA00,032),CUNUMBR=(FA00),UNIT=IQD
           IODEVICE ADDRESS=(FB00,032),CUNUMBR=(FB00),UNIT=IQD
```

Figure 16 HiperSockets IOCP statements

Figure 17 provides a graphical representation of this configuration.

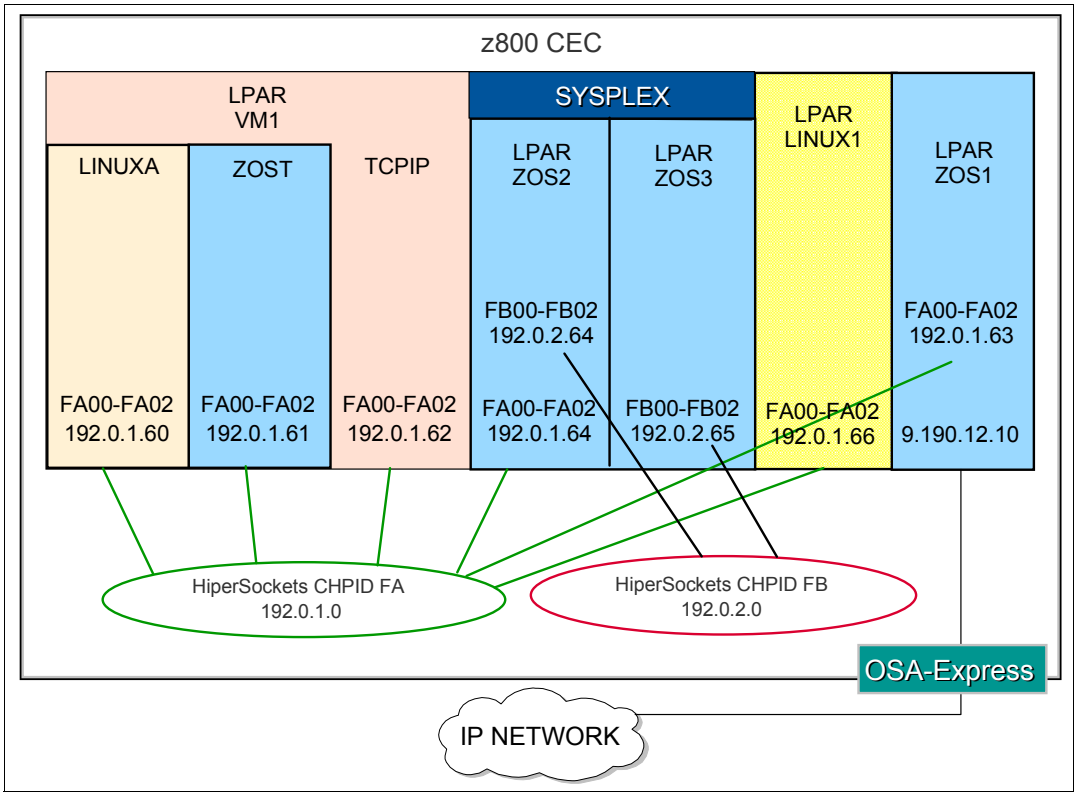

Figure 17 HiperSockets example environment

z/VM considerations

Each HiperSockets connection requires three I/O devices. One device is used for read control, one device is used for write control, and one device is used for data exchange. The device number for the control write device must be the device number for the read control device plus 1. The device number for the data exchange device can be any number.

We can use the VM DEDICATE command to reserve these devices for a particular Linux guest.

The syntax of the DEDICATE statement is:

```
DEDICATE virtual_address real_address
```

Figure 18 illustrates the DEDICATE statement usage in the directory entry for a Linux guest.

```
DEDICATE FA00 FA00
DEDICATE FA01 FA01
DEDICATE FA02 FA02
```

Figure 18 Extract from a user directory of a HiperSockets interface dedicated to a Linux guest

Using HiperSockets with Linux

From a Linux on zSeries perspective, the HiperSockets interface looks a lot like an OSA-Express (QDIO mode) interface. Linux uses the qdio and qeth modules to exploit HiperSockets.

An example /etc/chandev.conf configuration for a HiperSockets interface is as follows:

```
noauto;qeth-1,0xfa00,0xfa01,0xfa02;add_parms,0x10,0xfa00,0xfa02
```

Table 4 provides a description of the parameters.

Table 4 qeth device driver parameters

Parameter	Description
noauto	Stops auto-detection of channel devices.
noauto;qeth-1,0xfa00,0xfa01,0xfa02,0,0	
qeth-1	The device interface number. A value of “-1” indicates that the next available device number will be automatically allocated. Even though we used “qeth” as the device interface type, the actual interface name will start with “hsi”. So, for example, you would do an “ifconfig hsi0” to display the interface after the device drivers have been loaded.
0xfa00	The read subchannel address.
0xfa01	The write subchannel address.
0xfa02	The data subchannel address.
0,	The number of kilobytes to be allocated for read and write buffers. 0 specifies the default value (8192 KB in QDIO mode).
0	The relative port number of the CHPID. HiperSockets devices use only port 0.
add_parms,0x10,0xfa00,0xfa02	
add_parms	Used to pass additional parameters to the driver.
0x10	Identifies the device as an OSA-Express CHPID in QDIO mode.
0xfa00,0xfa02	The desired device address range.

Note: We do not need to use a PORTNAME when using HiperSockets.

HiperSockets Accelerator

The HiperSockets Accelerator (HSA) allows a z/OS TCP/IP stack acting as a router to route packets from an OSA-Express (QDIO mode) interface to a HiperSockets LAN. This can potentially reduce the number of direct OSA-Express connections that are required. External traffic is “concentrated” to a specific z/OS TCP/IP stack and then internally routed to other TCP/IP stacks in other LPARs.

The routing is done by the z/OS Communications Server device drivers at the lowest possible software data link control level. IP packets do not have to be processed at the higher-level TCP/IP stack routing function, thus reducing the path length and improving performance. As with point-to-point and z/VM Guest LAN solutions, the Linux systems behind the z/OS TCP/IP router must be on a different IP subnet from the external network.

HiperSockets Accelerator is illustrated in Figure 19. For a detailed review of HiperSockets Accelerator, refer to *zSeries HiperSockets*, SG24-6816.

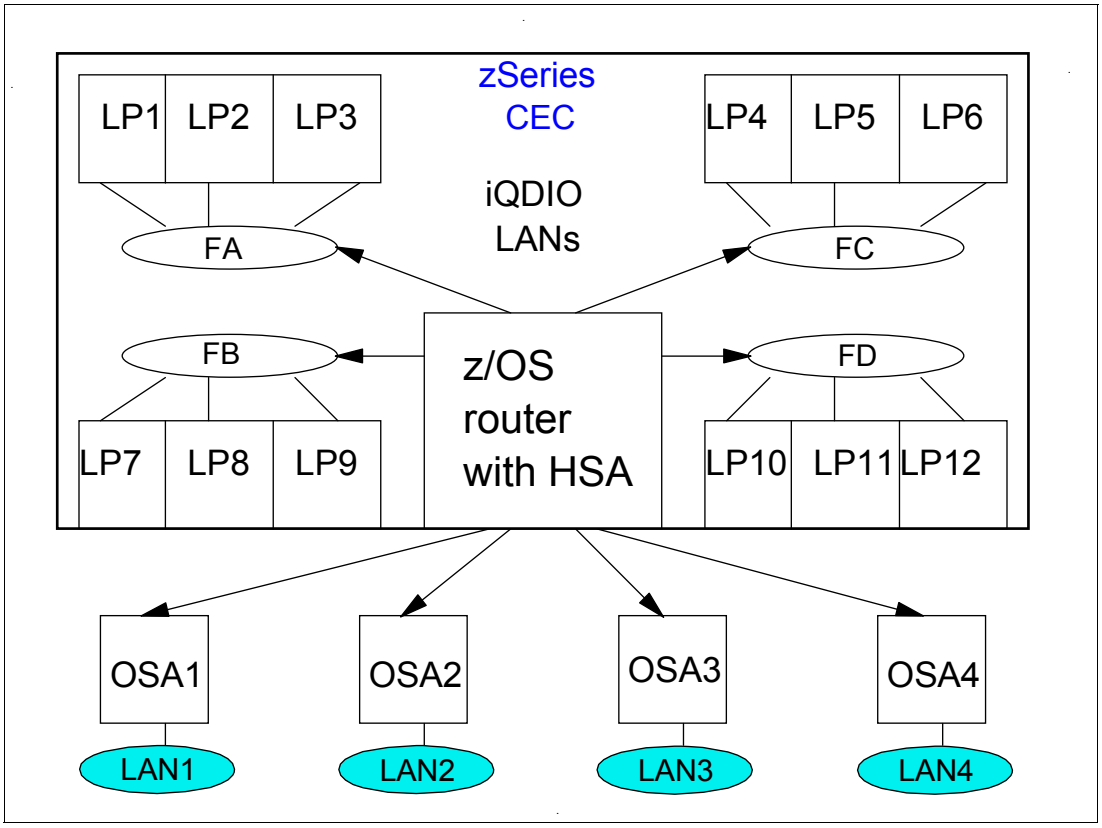

Figure 19 HiperSockets Accelerator

HiperSockets Network Concentrator

The HiperSockets Network Concentrator (depicted in Figure 20 on page 26) is a mechanism to connect systems that have HiperSockets interfaces to the external network using the same subnet. In other words, the Linux systems appear as though they are directly connected to the physical network. A Linux system acts as a forwarder for traffic between the OSA interface and the internal HiperSockets-connected systems. These systems do not have to run Linux. Refer to *Linux for zSeries and S/390, Device Drivers and Installation Commands, October 7, 2004*, LNUX-1313-04, for detailed information about how to configure a HiperSockets Network Concentrator. HiperSockets Network Concentrator can be a useful solution if you do not run z/VM V5.1 on a z890 or z990 and thus cannot implement Layer 2 Switching using the z/VM Virtual Switch (see for “Layer 2 LAN Switching” on page 53 more details).

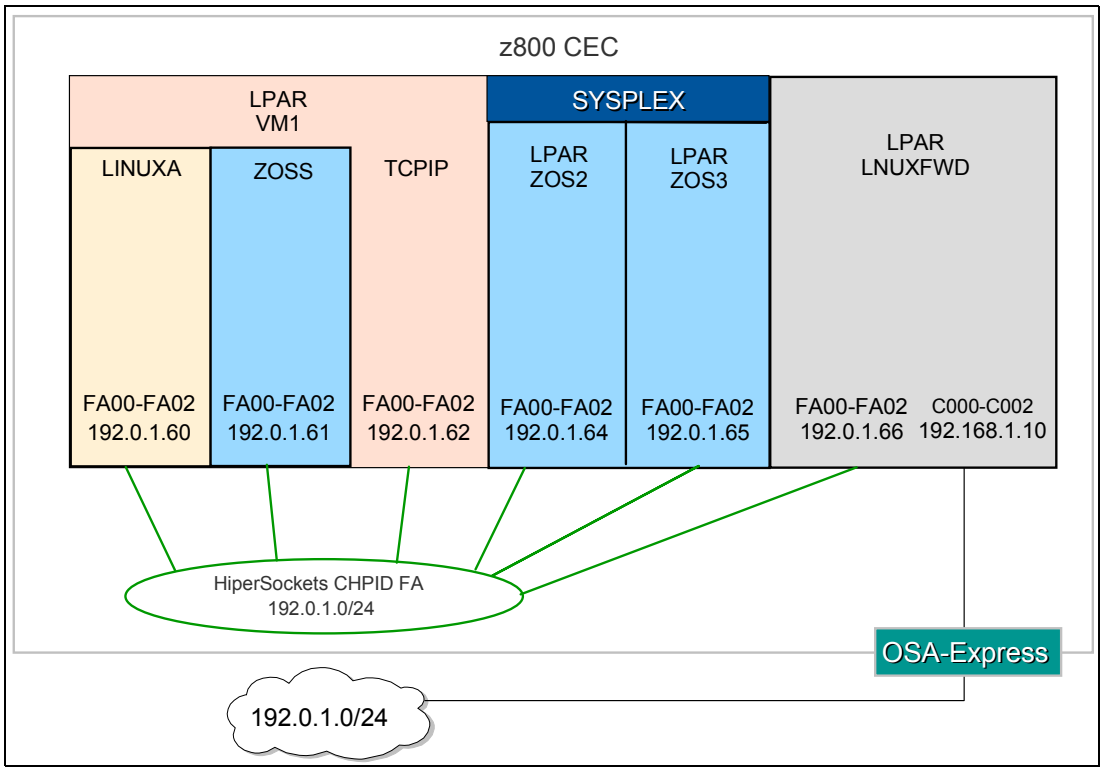

Figure 20 HiperSockets Network Concentrator example

Recommendations

HiperSockets is an excellent choice if you need to communicate across servers running in multiple LPARs in a single processor.

These servers can communicate at memory speeds, bypassing all the network overhead and delays. With HiperSockets, a maximum frame size can be defined according to the traffic characteristics for each of the HiperSockets LANs. By contrast, LANs such as Ethernet and token ring have a maximum frame size determined by their architecture.

Because there is no server-to-server traffic outside the zSeries CEC, a much higher level of network availability, security, simplicity, performance, and cost effectiveness is achieved as compared with servers communicating across a LAN. For example, because HiperSockets has no external components, it provides a very secure connection. For security purposes, servers can be connected to different HiperSockets LANs. All security features, such as firewall filtering, are available for HiperSockets interfaces as they are with other TCP/IP network interfaces. HiperSockets looks like any other TCP/IP interface; therefore, it is transparent to applications and supported operating systems.

If you need to communicate across different LPARs within a zSeries processor, we recommend that you use HiperSockets.

Virtualization technology

In a z/VM environment, the Linux operating system runs inside a virtual machine (*VM Guest*). As well as directly connecting to a physical interface, such as an OSA-Express card, the Linux guest can use a virtual interface, which is a VM control program simulation of a physical interface. These virtual interfaces can be broadly split into two categories, point-to-point connectivity and VM simulated LAN technology.

Point-to-point connectivity

Prior to z/VM V4.2, the virtual connectivity options for connecting one or more virtual machines were limited to virtual channel-to-channel adapters (CTCA) and the Inter-User Communications Vehicle (IUCV) facility. These virtual interfaces are classified as point-to-point connections.

Using point-to-point connectivity, the z/VM TCP/IP stack or a Linux guest has to act as a router between the external network and the Linux guests. This means that additional routing definitions are required in the network so that other machines know that, in order to communicate with the Linux guests, they must go through the z/VM TCP/IP or Linux router. Although the bandwidth of point-to-point connections is considerable and thus affords the rapid movement of large amounts of data between guests, these interfaces have a number of drawbacks.

Using CTCA links as an example, in order for a Linux guest to communicate with the external network, you must define CTCA device pairs on both the Linux and VM TCP/IP side. If you also have a requirement for individual Linux guests to communicate with each other, you also need to configure additional CTCA devices for those links (see the connections between Linux1 and Linux 2 in Figure 21 on page 28). CTCA devices on both sides of the connection then need to be coupled together. Static routing statements must be defined in both the Linux and VM TCP/IP stacks. Finally, if one side of the point-to-point connection goes down, it is often difficult to subsequently reconnect the two guests. Frequently, one of the Linux guest machines has to be rebooted in order to reestablish the connection.

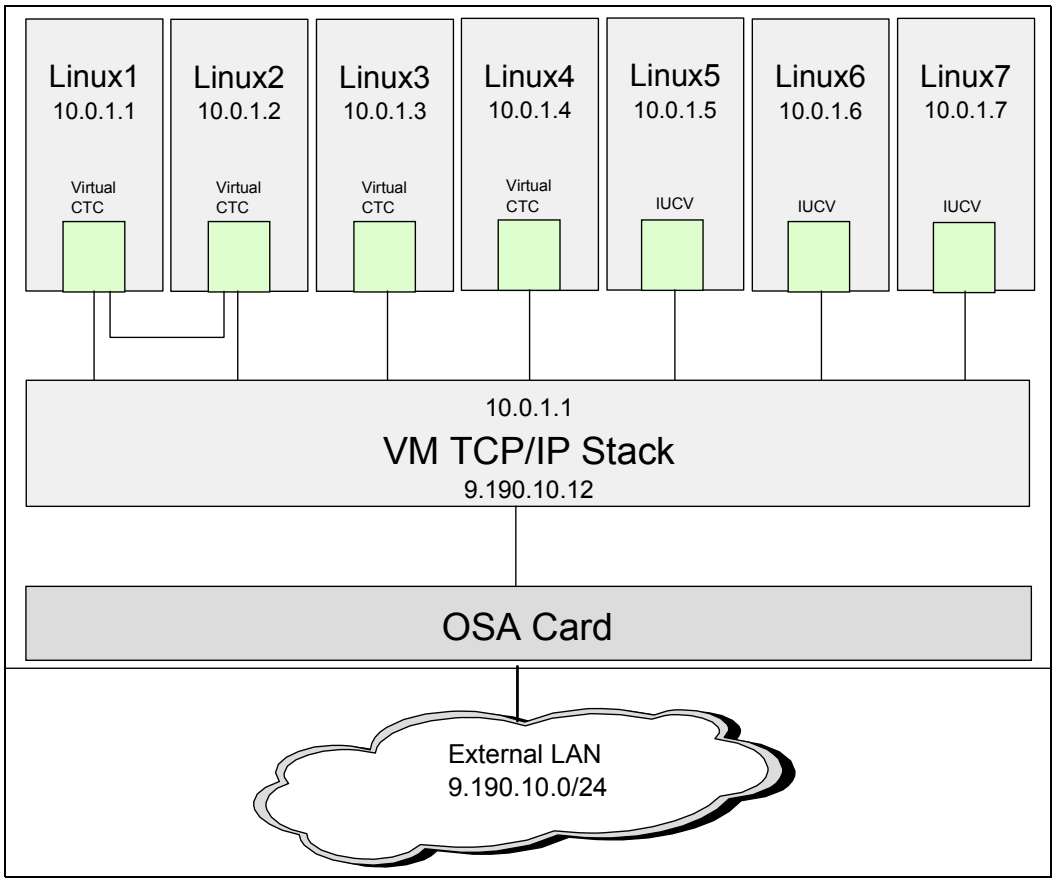

Figure 21 VM point-to-point connections

Recommendations

Given the complexity of managing a point-to-point network within a z/VM system, we do not recommend it as a solution. When it was the only topology we had available to communicate among guests, obviously, it was acceptable. We now recommend that you use one of the z/VM LAN technologies, Guest LAN or VSWITCH.

Guest LAN

From z/VM V4.2 and later, the z/VM control program (CP) has been enhanced to provide a feature known as *Guest LAN*. This feature enables you to create multiple Virtual LAN segments within a z/VM environment. As can be seen from Figure 22 on page 29, Guest LANs do not have a physical connection to the external network. Instead, they must use a router (z/VM TCP/IP or Linux) in the same fashion as was required for the “point-to-point” topology. The Linux router (or z/VM TCP/IP stack) must have an external interface, such as an OSA-Express card, and an interface connected to the Guest LAN.

Note: Although the structures and simulated devices related to the Guest LAN under z/VM are “virtual,” we use the term *Guest LAN* and not *Virtual LAN*, because the term Virtual LAN (VLAN) has a different meaning in the networking world.

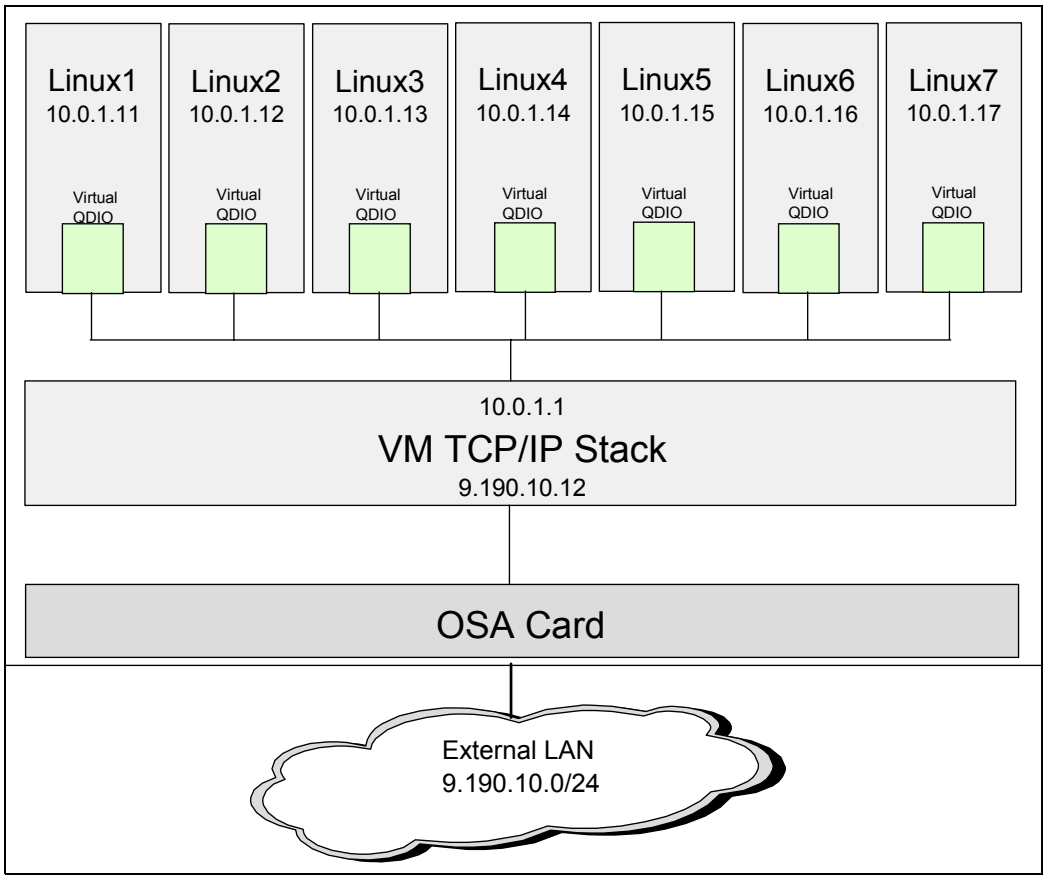

Figure 22 z/VM Guest LAN

There is no architectural limit on the number of Guest LAN segments that can be created. The actual limit will be governed by available machine resources, but this is more of a theoretical limit than a practical one. The number of Guest LANs that can be created can be considered unlimited for all practical purposes.

In contrast to point-to-point connections, to connect to a Guest LAN, individual guest machines create a virtual Network Interface Card (NIC). They can connect this NIC to the Guest LAN and communicate with other guests using standard TCP/IP protocols. Defining links to individual guests and coding static routes for each link are no longer required, because the guests exist in the same Guest LAN segment.

Note: A z/VM Guest LAN is contained solely within a z/VM LPAR. A Linux guest connected to that Guest LAN cannot directly communicate (using its Guest LAN interface) with hosts outside of the Guest LAN.

As z/VM Guest LAN is a virtualization technology, it is not limited to the use of zSeries hardware. Guest LAN is supported on all IBM mainframe hardware from IBM 9672 Generation 5 processors onward (including the Multiprise® 3000).

When Guest LAN was first released with z/VM V4.2, the virtual NIC simulated a HiperSockets device, as introduced on the zSeries 900.

From z/VM V4.3, the Guest LAN could be defined to use either simulated HiperSockets (iQDIO, or internal QDIO) devices or simulated QDIO devices. The QDIO device that is

simulated is the OSA-Express Ethernet adapter. All examples in this Redpaper use the QDIO Guest LAN and not the HiperSockets Guest LAN.

Note: Even though QDIO and HiperSockets Guest LANs are different, when it comes to defining the virtual Network Interface Card to the Linux guest, they both use the same qdio and qeth device drivers. However, QDIO Guest LANs use a device identifier of ethx, and HiperSockets Guest LANs use a device identifier of hsix.

QDIO versus iQDIO

When deciding whether to deploy QDIO Guest LAN or HiperSockets (iQDIO) Guest LAN, consider the following information:

- QDIO (OSA-Express simulation)
  - IPv4 and IPv6 support
  - Easy to migrate from QDIO Guest LAN to VSWITCH
  - Ethernet transport
  - Can be used as an OSA-Express test network
- iQDIO (HiperSockets simulation)
  - IPv4 support
  - Supports multicast router connections
  - Deploy MTUs larger than 8 K
  - Can be used as a HiperSockets test network
  - Slightly smaller path length in CP than QDIO Guest LAN

Guest LAN configuration

To create a Guest LAN, use the following required steps:

1. Create a z/VM Guest LAN segment in the z/VM host system.
2. Create a virtual Network Interface Card (NIC) in each guest machine that will be connected to the Guest LAN.
3. Connect the virtual NIC in each guest machine to the Guest LAN.
4. After the Linux guest has been booted, configure the appropriate device drivers in that guest to connect to the Guest LAN.

Tip: Although the syntax in the following sections is valid, we recommend that you always refer to the relevant level of z/VM reference manual for a complete description of all command syntax. For the following examples, we used *z/VM Version 5 Release 1.0 CP Planning and Administration*, SC24-6083, and *z/VM Version 5 Release 1.0 CP Commands and Utility Reference*, SC24-6081.

Create a z/VM Guest LAN

z/VM Guest LANs can be created in one of two ways: either by a DEFINE LAN statement in the z/VM CP SYSTEM CONFIG file or by using the DEFINE LAN CP command. Why choose one over the other? Guest LANs created with the DEFINE LAN command are only valid for the life of a z/VM system. In other words, if that system is shut down and then IPLed, the Guest LAN will no longer be defined. We recommend that for production Guest LANs you

make a permanent entry in the CP SYSTEM CONFIG file. This is known as a *persistent* Guest LAN. For testing purposes, the DEFINE LAN command is perfectly valid because it provides the flexibility to dynamically create Guest LANs as required.

Persistent Guest LANs

In order to define a persistent Guest LAN, we need to add a DEFINE LAN statement to the CP SYSTEM CONFIG file. For a complete discussion of making changes to this file, refer to the relevant level of the z/VM reference manual.

The syntax of the DEFINE LAN statement is as follows:

```
DEFINE LAN lanname [ operands ]
```

Where:

lanname Is a 1-8 alphanumeric name of the z/VM Guest LAN.

operands Define the characteristics of the z/VM Guest LAN.

Table 5 summarizes the operands accepted by the DEFINE LAN statement.

Table 5 Operands of the DEFINE LAN statement

Operand	Description
OWNERid *ownerid*	Establishes the owner of the LAN. When OWNERid * is specified, the owner is the invoker. In our examples, the ownerid will be SYSTEM.
TYPE *lantype*	Specifies the type of LAN. Valid types are HIPERsockets for simulated HiperSockets adapters or QDIO for simulated QDIO adapters. HiperSockets is the default.
IP | ETHernet	For QDIO Guest LANs, this indicates whether the transport for the LAN is Ethernet or IP. An Ethernet LAN operates at the Layer 2 level of the OSI model. An IP LAN operates at Layer 3 of the OSI model.
MAXCONN *maxconn*	Sets the maximum number of simultaneous adapter connections permitted. When MAXCONN is specified as INFinite, there is no limit on the number of connections. Any other value limits the number of simultaneous connections to a decimal value in the range of 1-1024.
MFS *size*	Sets the Maximum Frame Size (MFS) for adapters on this network. When an adapter is connected to this LAN, it will adopt the network MFS. The MFS value determines the amount of storage to be allocated for internal structures and limits the effective Maximum Transfer Unit (MTU) size for the coupled adapters. The MFS operand is not valid for the QDIO Guest LAN; however, the effective MFS is 8 K for a QDIO adapter.
UNRESTricted	Defines a LAN with no access control; therefore, any user can connect to the LAN. When neither UNRESTricted nor RESTricted are specified, UNRESTricted is the default value.
RESTricted	Defines a LAN with an access list to restrict connections. The LAN owner will use the SET LAN command to grant or revoke access to specific VM users (by user ID). The COUPLE command will only allow authorized users (those on the access list) to connect a simulated adapter to a RESTRICTED network.
ACCOUNTING *value*	Allows a Class B user to control whether accounting records are created for the LAN being defined.
GRANT *userlist*	Defines the list of users to be included in the Initial Access List of a RESTRICTED LAN. If the GRANT operand is omitted, the default is to GRANT the LAN owner.

As an example, to create a QDIO type Guest LAN named TSTLAN owned by SYSTEM, use:

```
DEFINE LAN TSTLAN OWNERID SYSTEM TYPE QDIO
```

The VMLAN statement

In addition to the DEFINE LAN statement, we can also add a VMLAN statement to the CP SYSTEM CONFIG file to establish system-wide attributes for all z/VM Guest LANs that have been defined to the z/VM operating system.

The syntax of the **VMLAN** statement is as follows:

```
VMLAN LIMIT [ operands ]
```

Or:

```
VMLAN ACCOUNTing [ operands ]
```

Where *operands* define the attributes to be set for all z/VM Guest LANs in the system.

Table 6 summarizes the operands accepted by the VMLAN command.

Table 6 Operands of the VMLAN statement

LIMIT parameter operands	Description
PERSistent INFinite | *maxcount*	INFinite means that there will be an infinite number of PERSISTENT z/VM Guest LAN segments allowed on the system. INFinite is the default. Use the *maxcount* parameter to define a number of PERSISTENT Guest LANS (between 0 and 1024) permitted to run on the system.
TRANSient INFinite | *maxcount*	INFinite means that there will be an infinite number of TRANSIENT z/VM Guest LAN segments allowed on the system. INFinite is the default. Use the *maxcount* parameter to define a number of TRANSIENT Guest LANS (between 0 and 1024) permitted to run on the system.
ACNT | ACCOUNTing SYSTEM ON | OFF	Set the default accounting state for z/VM Guest LAN segments owned by the SYSTEM user ID. The default state of this attribute is OFF.
ACNT | ACCOUNTing USER ON | OFF	Set the default accounting state for z/VM Guest LAN segments owned by individual users. The default state of this attribute is OFF.
MACPREFIX *macprefix*	Specifies the 3-byte prefix (manufacturer ID) used when generating locally administered MAC addresses on the system. It must be six hexadecimal digits within the range of 020000 through 02FFFF (inclusive). In combination with the MAC ID used on the NICDEF directory statement, the MACPREFIX allows unique identification of virtual adapters within a network. If MACPREFIX is not specified, the default is 020000 02-00-00).
MACIDRange SYSTEM *xxxxxx-xxxxxx* USER *xxxxxx-xxxxxx*	

LIMIT parameter operands	Description
MACIDRange SYSTEM *xxxxxx-xxxxxx*	The range of identifiers (up to six hexadecimal digits each) to be used by CP when generating the unique identifier part (last six hexadecimal digits) of a virtual adapter MAC address. If a SYSTEM MACIDRANGE is not specified, CP creates unique identifiers in any range (000001-FFFFFF).
USER *xxxxxx-xxxxxx*	USER *xxxxxx-xxxxxx* is the subset of the SYSTEM range of identifiers that are reserved for user definition of MACIDs in the NICDEF directory statement. When specified, CP does not assign MACIDs within this USER range during creation of virtual adapters defined dynamically (DEFINE NIC) or with the NICDEF (or SPECIAL) directory statement without the MACID operand. In these cases, CP generates a unique identifier for the adapter outside of the USER range. Any MACID values specified on a NICDEF directory statement must be within the USER range, or the virtual adapter is not defined during LOGON processing. If a USER MACIDRANGE is not specified, CP creates unique identifiers within the SYSTEM MACIDRANGE.

Create a virtual Network Interface Card

You must create a virtual Network Interface Card (NIC) for each guest machine. Once defined, this NIC can be connected to the Guest LAN. To the guest operating system, the NIC devices look like a range of OSA devices. The NIC can be defined permanently through a User Directory statement or temporarily (for the life of the Guest's session) through a CP command.

NIC definition in the user directory

To create a virtual Network Interface Card that will remain permanently defined to a VM guest machine (that is, across guest sessions and across IPLs of the z/VM operating system), use the NICDEF statement in the z/VM User Directory. The NICDEF statement defines virtual devices that are fully simulated by CP. The NIC automatically joins the Guest LAN when the z/VM user ID is logged on. The syntax of the NICDEF statement for Network Interface Cards is as follows:

```
NICDEF vdev [ operands ]
```

Where:

vdev Specifies the base virtual device address for the adapter.

operands Define the characteristics of the virtual NIC.

Table 7 lists the operands accepted by the NICEF command.

Table 7 Operands for the NICDEF user directory statement

Operands	Description
HIPERs | QDIO	HIPERs indicates that a simulated HiperSockets adapter should be created. QDIO indicates that a simulated QDIO adapter should be created. If a LAN is identified in this statement or another with the same *vdev*, the NIC is automatically coupled to the specified *ownerid lanname*.
devs	The number (decimal) of virtual I/O devices to be created for a simulated NIC. If devs is omitted, the default number of devices is three.
ownerid | ** lanname*	Identifies a Guest LAN segment for an immediate connection to the NIC. If ownerid and lanname are omitted, the simulated adapter is left in the uncoupled state. When ownerid and lanname are specified, the adapter is automatically connected to the designated Guest LAN. Note that the ownerid can be specified as a name or using an asterisk (*) to represent the user ID of the current virtual machine.
CHPID *xx*	A two-digit hexadecimal number that represents the CHPID number to be allocated in the virtual machine I/O configuration for this adapter. If CHPID is omitted, an available CHPID is automatically assigned to this adapter. This option is required when a HiperSockets adapter is being created for a z/OS guest, because z/OS configurations require a predictable CHPID number. During LOGON, CP attempts to use the specified CHPID number. If the specified CHPID number is already in use, this adapter is not defined. To correct this situation, you must eliminate the conflicting device or select a different CHPID.
MACID *xxxxxx*	A unique identifier (up to six hexadecimal digits) used as part of the adapter MAC address. During LOGON, your MACID (3 bytes) is appended to the system MACPREFIX (3 bytes) to form a unique MAC address for this adapter. If MACID is omitted from this definition, CP generates a unique identifier for this adapter. If the specified MACID is already in use, this adapter is not defined. To correct this situation, you must eliminate the conflicting device or select a different MACID.

Figure 23 shows an example CP User Directory entry for a Linux guest that connects to a QDIO Guest LAN.

```
USER LNX23 LNX23 128M 1G G
   INCLUDE IBMDFLT
   IPL 190 PARM AUTOCR
   MACHINE XA
   CONSOLE 0009 3215
   NICDEF 0700 TYPE QDIO DEV 3 SYSTEM TSTLAN
   MDISK 0191 3390 3274 025 LEVW01 MR READ WRITE MULTIPLE
   MDISK 0201 3390 3339 0200 LX3EA3 MR READ WRITE MULTIPLE
   MDISK 0202 3390 3539 3138 LX3EA3 MR READ WRITE MULTIPLE
```

Figure 23 User directory entry for a Linux guest: Connecting to a QDIO Guest LAN

NIC definition using CP commands

To create a virtual Network Interface Card that will only last for the life of a guest (that is, it will need to be redefined when the guest next logs on to the system), use the following command syntax:

```
DEFINE NIC vdev [ operands ]
```

Where:

vdev Specifies the base virtual device address for the adapter.

operands Define the characteristics of the virtual NIC.

Table 8 lists the operands accepted by the DEFINE NIC command.

Table 8 Operands for the DEFINE NIC command

Operands	Description
HIPERsockets	Defines this adapter as a simulated HiperSockets NIC. This adapter will function like the HiperSockets internal adapter. A HiperSockets NIC can function without a z/VM Guest LAN connection, or it can be coupled to a HiperSockets Guest LAN.
QDIO	Defines this adapter as a simulated QDIO NIC. This adapter will function like the OSA-Express (QDIO) adapter. A QDIO NIC is only functional when it is coupled to a QDIO Guest LAN.
DEVices *devs*	Determines the number of virtual devices associated with this adapter. For a simulated HiperSockets adapter, devs must be a decimal value between 3 and 3072 (inclusive). For a simulated QDIO adapter, devs must be a decimal value between 3 and 240 (inclusive). The DEFINE NIC command will create a range of virtual devices from vdev to vdev + devs -1 to represent this adapter in your virtual machine. The default value is 3.
CHPID *nn*	A two-digit hexadecimal number that represents the CHPID number the invoker wants to allocate for this simulated adapter. If the requested CHPID number is available, all of the virtual devices belonging to this adapter will share the same CHPID number. This option is only useful if you need to configure a virtual environment with predictable CHPID numbers for your simulated devices.

Connect the virtual NIC to the Guest LAN

Now that we have defined the virtual NIC, just as in a real network, we need to connect that device to the LAN. If we had used the NICDEF User Directory statement to define our NIC, the guest machine would automatically connect to the LAN whenever it logged on. However, if we chose to use the DEFINE NIC command, we have an additional step to perform before the device is connected to the Guest LAN.

Use the COUPLE CP command to attach the virtual NIC to a compatible Guest LAN. The syntax of the COUPLE command for this scenario is:

```
COUPLE vdev TO [ operands ]
```

Where:

vdev Specifies the base virtual device address for the adapter.

operands Defines where to connect the NIC.

Table 9 on page 36 lists the operands accepted by the COUPLE command for the purpose of connecting a virtual NIC to a Guest LAN.

Table 9 Operands for the COUPLE command

Operands	Description
vdev	The base address of the network adapter.
ownerid lanname	The ownerid is the name of the owner of the Guest LAN (for example, SYSTEM). The lanname is the name of the Guest LAN.

Remember that a virtual NIC can only be coupled to a *compatible* Guest LAN. For example, a QDIO NIC cannot be coupled to a Guest LAN of type “HIPERsockets.”

Tip: If you choose to use the DEFINE NIC and COUPLE approach instead of the NICDEF User Directory statement, consider adding these two commands into your guest’s PROFILE EXEC file so that they are automatically executed whenever the guest logs on.

Example of building a z/VM Guest LAN

We now demonstrate how to build a z/VM Guest LAN in Figure 24.

```
DEFINE LAN TSTLAN OWNERID SYSTEM TYPE QDIO   1
LAN SYSTEM TSTLAN is created
Ready; T=0.01/0.01 17:17:30

DEFINE NIC 0700 QDIO                         2
NIC 0700 is created; devices 0700-0702 defined
Ready; T=0.01/0.01 17:22:40

COUPLE 0700 TO SYSTEM TSTLAN                 3
NIC 0700 is connected to LAN SYSTEM TSTLAN
Ready; T=0.01/0.01 17:23:04
```

Figure 24 Steps to build a z/VM Guest LAN

To build a z/VM Guest LAN:

1. Define a QDIO Guest LAN owned by SYSTEM. This command was run from the MAINT user.
2. Define a Network Interface Card (NIC) of type QDIO. This command was run from Linux guest user LNX23.
3. Couple the NIC to the Guest LAN. This command was also run from LNX23.

Now that we have built a Guest LAN, we can use the CP QUERY LAN command to verify the status of the LAN, as shown in Figure 25.

```
QUERY LAN TSTLAN ACTIVE
LAN SYSTEM TSTLAN      Type: QDIO     Active: 1     MAXCONN: INFINITE
  PERSISTENT  UNRESTRICTED  MFS: 8192    ACCOUNTING: OFF
Ready; T=0.01/0.01 17:36:49
```

Figure 25 CP QUERY LAN command

In order to display information about the virtual NIC that we have defined, we can use the QUERY NIC CP command, as shown in Figure 26 on page 37. If we use the DETAILS parameter of this command, we can get additional information about the IP addresses bound

to this NIC and the amount of data that has been transmitted and received through this interface (TX packets/bytes and RX packets/bytes, respectively).

Notice that there is no IP addressing information and the number of bytes transmitted and received are both zero. Also, the port name value is set to UNASSIGNED. This tells us that the Linux guest has not started using this device for TCP/IP communications.

```
QUERY NIC
Adapter 0700  Type: QDIO      Name: UNASSIGNED  Devices: 3
  Port 0 MAC: 02-00-00-00-00-06  LAN: SYSTEM TSTLAN        MFS: 8192

Q NIC DETAILS
Adapter 0700  Type: QDIO      Name: UNASSIGNED  Devices: 3
  Port 0 MAC: 02-00-00-00-00-06  LAN: SYSTEM TSTLAN        MFS: 8192
  RX Packets: 0          Discarded: 0          Errors: 0
  TX Packets: 0          Discarded: 0          Errors: 0
  RX Bytes: 0                    TX Bytes: 0
  Unassigned Devices:
      Device: 0700  Unit: 000   Role: Unassigned
      Device: 0701  Unit: 001   Role: Unassigned
      Device: 0702  Unit: 002   Role: Unassigned
```

Figure 26 CP QUERY NIC commands

Finally, we said that the virtual NIC simulates an OSA-Express QDIO device. This is confirmed by using the CP command QUERY VIRTUAL OSA from the guest machine, as shown in Figure 27.

```
Q VIRTUAL OSA
OSA  0700 ON NIC  0700  UNIT 000 SUBCHANNEL = 0010
     0700 QDIO-ELIGIBLE       QIOASSIST NOT AVAILABLE
OSA  0701 ON NIC  0700  UNIT 001 SUBCHANNEL = 0011
     0701 QDIO-ELIGIBLE       QIOASSIST NOT AVAILABLE
OSA  0702 ON NIC  0700  UNIT 002 SUBCHANNEL = 0012
     0702 QDIO-ELIGIBLE       QIOASSIST NOT AVAILABLE
```

Figure 27 CP QUERY VIRTUAL OSA command

Undoing the definitions

Before moving on to describe how to connect a Linux guest to the Guest LAN, we review how to undo the previous definitions in an orderly fashion. This is only for completeness, and you should not follow these steps unless you no longer want to use the z/VM Guest LAN that you created.

Disconnect from a Guest LAN

The CP UNCOUPLE command is used to disconnect a virtual NIC from a Guest LAN segment. Figure 28 illustrates this command.

```
UNCOUPLE 700
NIC 0700 is disconnected from LAN SYSTEM TSTLAN
Ready; T=0.01/0.01 18:37:15
```

Figure 28 UNCOUPLE command

Remove the virtual NIC from the guest machine

To remove a virtual NIC from a guest machine, use the CP DETACH NIC command. The command disconnects the virtual adapter from the Guest LAN (assuming the UNCOUPLE command has not been invoked) and removes each virtual device that has been created. Figure 29 illustrates the DETACH NIC command.

```
DETACH NIC 0700
NIC 0700 is destroyed; devices 0700-0702 detached
Ready; T=0.01/0.01 18:40:39
```

Figure 29 DETACH NIC command

Remove the Guest LAN

To remove a Guest LAN from the system, use the CP DETACH LAN command. This command removes the LAN from the System LAN table, disconnects any virtual adapters that were using the LAN, and releases system resources associated with the LAN. Figure 30 illustrates the DETACH LAN command.

```
DETACH LAN TSTLAN OWNER SYSTEM
LAN SYSTEM TSTLAN is destroyed
Ready; T=0.01/0.01 18:44:29
```

Figure 30 DETACH LAN command

Configuring Linux to connect to a Guest LAN

Now that we have created a z/VM Guest LAN and connected our guest's virtual NIC to that LAN, the final step is to boot the Linux operating system and load the qeth device driver for the virtual Network Interface Card. This is the same device driver we use for the OSA-Express card.

Before we load the qeth driver, we must first pass configuration information about the virtual NIC to the Linux kernel's channel device layer. See "Channel device layer" on page 6.

Figure 31 shows the configuration settings we redirected to the /proc/chandev file.

```
echo 'noauto ' >>/proc/chandev
echo 'qeth-1,0x0700,0x0701,0x0702,0,0 '>>/proc/chandev
echo 'add_parms,0x10,0x0700,0x0702,portname:TSTLAN ' >>/proc/chandev
```

Figure 31 Redirecting qeth device driver settings to /proc/chandev

Table 10 provides a description each of these settings.

Table 10 qeth device driver parameters

Parameter	Description
`noauto`	Stops auto-detection of channel devices.
`qeth-1,0x0700,0x0701,0x0702,0,0`	
`qeth-1`	The device interface number. A value of "-1" indicates that the next available device number will be automatically allocated. For example, if we already had qeth0 and qeth1 devices defined to the channel device layer, the next device to be defined would be qeth2.
`0x700`	The read subchannel address.
`0x701`	The write subchannel address.
`0x702`	The data subchannel address.
`0,`	The number of kilobytes to be allocated for read and write buffers. 0 specifies the default value (8192 KB in QDIO mode).
`0`	The relative port number of the CHPID. OSA-Express devices (including our simulated QDIO NIC) use only port 0.
`add_parms,0x10,0x0700,0x0702,portname:TSTLAN`	
`add_parms`	Used to pass additional parameters to the driver.
`0x10`	Identifies the device as an OSA-Express CHPID in QDIO mode.
`0x700,0x702`	The desired device address range.
`portname:TSTLAN`	Identifies the port name as TSTLAN. This is the name of our z/VM Guest LAN.

Tip: Strictly speaking, the port name parameter (in our case, TSTLAN) is not required for z/VM Guest LANs. However, we have chosen to use the name of the Guest LAN for the port name as a means of documenting to which Guest LAN this interface is connected.

Figure 32 on page 40 shows a partial extract from the /proc/chandev file, showing only the changes after we redirected our settings for the virtual NIC.

```
No auto devno ranges                                                    1
   From        To
====================
  0x0000     0xffff

Forced devices
chan defif read   write  data   memory      port         ip    hw   host
type  num  devno  devno  devno  usage(k) protocol no.  chksum stats name
================================================================================
0x10    0  0x0700 0x0701 0x0702 default         0       0    0


Initialised Devices
 read   write  data  read   write  data  chan port  dev     dev
 irq     irq    irq  devno  devno  devno type no.   ptr     name
================================================================================
0x0011 0x0012 0x0013 0x0700 0x0701 0x0702 0x10  0 0x04467200 eth0      8192 2


channels detected
              chan    cu    cu   dev    dev                          in chandev
  irq  devno  type   type  model type  model pim      chpids         use  reg.
================================================================================
0x0011 0x0700 0x10  0x1731 0x01  0x1732 0x01 0x80 0x0600000000000000  yes  yes
0x0012 0x0701 0x10  0x1731 0x01  0x1732 0x01 0x80 0x0600000000000000  yes  yes
0x0013 0x0702 0x10  0x1731 0x01  0x1732 0x01 0x80 0x0600000000000000  yes  yes

 driver specific parameters
chan    lo    hi      driver
type  devno  devno  parameters
================================================================================
0x10 0x0700 0x0702  portname:TSTLAN,                                        3
```

Figure 32 Partial contents of /proc/chandev after redirecting the qeth device driver settings

The changes after redirecting the qeth device driver settings include:

- We chose to use noauto. Because we did not specify any devices, auto-detection has been turned off for all devices.
- Our three devices, 0x700, 0x701, and 0x702, have been initialized. They are of channel type 0x10, meaning that they are defined as an OSA-Express QDIO style device, and the device name as it will appear to the TCP/IP stack is eth0.
- The z/VM Guest LAN that we are connected to has a port name called TSTLAN.

Loading the device driver

Now that the channel device layer contains our virtual NIC definitions, we can activate the virtual NIC device. To do that, we must load the appropriate device drivers. These device drivers are called *qdio* and *qeth*. The qdio driver is the underlying controlling driver for all devices that use the QDIO architecture. The qeth driver is specifically for our virtual NIC or for a real OSA-Express card or HiperSockets interface. In Linux, these device drivers are implemented as kernel modules.

First, we determine if the qdio and qeth device driver modules are already loaded using the **lsmod** command, as shown in Figure 33 on page 41.

```
# lsmod
8021q                  15256   1
qeth                  167996   1
ipv6                  329288  -1  [qeth]
qdio                   37040   2  [qeth]
```

Figure 33 The lsmod command output

We see the modules are loaded; this is because we were using an OSA-Express card for connectivity to our test Linux guest. The 8021q and ipv6 modules provide support for VLAN and IPv6, respectively.

Important: If the Linux system has the qdio and qeth modules already loaded, you would not want to unload and reload those modules just to enable the virtual NIC device. Doing so would disrupt connectivity to the devices that rely on those modules. Instead, after you have redirected the settings for the virtual NIC to the channel device layer, you can activate the new device by using the following command:

```
echo reprobe >/proc/chandev
```

The system will detect uninitialized channel devices and make them available for use.

Assuming the qdio and qeth modules were not loaded, we could use the following syntax to load these drivers:

```
insmod qdio
insmod qeth
```

Figure 34 on page 42 shows an example illustrating the output on the Linux guest's 3270 session when loading the device drivers.

```
insmod qdio
Using /lib/modules/2.4.19/kernel/drivers/s390/qdio.o
qdio: loading QDIO base support version 2 ($Revision: 1.120.2.1 $/$Revision: 1.5
6.2.1 $)
debug: qdio_setup: new level 2
debug: qdio_labs: new level 2
debug: qdio_sense: new level 2
debug: qdio_trace: new level 2

insmod qeth
Using /lib/modules/2.4.19/kernel/net/qeth.o
qeth: loading qeth S/390 OSA-Express driver ($Revision: 1.260.2.10d$/$Revision:
1.86.2.1 $/$Revision: 1.31 $:IPv6)
 qeth: allocated 0 spare buffers
debug: qeth_setup: new level 3
debug: qeth_misc: new level 2
debug: qeth_data: new level 2
debug: qeth_control: new level 2
debug: qeth_sense: new level 2
debug: qeth_qerr: new level 2
debug: qeth_trace: new level 2
qeth: Trying to use card with devnos 0x700/0x701/0x702
qeth: Device 0x700/0x701/0x702 is an OSD Express card (level: 2938)
with link type Gigabit Eth (portname: TSTLAN)
 qeth: IPv6 not supported on eth0
```

Figure 34 Load the qdio and qeth device drivers

Tip: For completeness, our example used the **insmod** command to insert the two modules. To make things easier, we could have used the **modprobe** command. The **modprobe** command will determine what dependencies a module has to other modules and also automatically load those other modules. So in our example, a **modprobe qeth** command would load both the qeth and qdio modules.

If we now run a CP QUERY NIC DETAILS command from the Linux guest's 3270 session, we see in Figure 35 that the NIC has now been linked to the Guest LAN at both a CP and Linux device driver level.

```
Q NIC DET
Adapter 0700  Type: QDIO      Name: TSTLAN  1 Devices: 3
  Port 0 MAC: 00-04-AC-00-00-01  LAN: SYSTEM TSTLAN      MFS: 8192
  RX Packets: 0          Discarded: 0          Errors: 0
  TX Packets: 0          Discarded: 0          Errors: 0
  RX Bytes: 0                   TX Bytes: 0
  Connection Name: HALLOLE   State: Session Established
    Device: 0700  Unit: 000   Role: CTL-READ
    Device: 0701  Unit: 001   Role: CTL-WRITE
    Device: 0702  Unit: 002   Role: DATA
```

Figure 35 QUERY NIC DETAILS example

The Name field is the port name as assigned by the channel device layer. Prior to loading the device drivers, the value of this field was set to UNASSIGNED. After making our changes to the /proc/chandev files and loading the device drivers, we see that we now have a value of TSTLAN, which matches the name of our Guest LAN.

If everything has worked correctly, you should now be able to run the **ifconfig -a** command to check that the virtual NIC is now available for use. Figure 36 shows the output of this command.

```
eth0      Link encap:Ethernet  HWaddr 00:04:AC:00:00:01
          MULTICAST  MTU:1492  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:17
```

Figure 36 Output of the ifconfig -a command

We could now use the **ifconfig** command to configure an IP address so that the guest could communicate with other machines on the Guest LAN. Alternatively, you could use a DHCP client to lease an address from a DHCP server machine that is running on the Guest LAN. For more information about using DHCP, refer to *Linux on IBM @server zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP-3596.[4]

You should now make the changes permanent by adding the configuration settings into the /etc/chandev.conf file. An extract of that file for our new virtual NIC is:

```
qeth0,0x0700,0x0701,0x0702,0,0;add_parms,0x10,0x0700,0x702,portname:TSTLAN
```

Table 10 on page 39 provides an explanation of the parameters.

Important: Before rebooting your Linux guest, you should first check the /etc/modules.conf file to verify that there is an `alias` entry in the file for your new network interface. For example, if you have just defined an interface called eth0, be sure that there is a matching `alias` statement. The kernel will look for a module called eth0, and when it cannot find it, it will check to see if eth0 is actually an alias for another module. In our case, eth0 is really the qeth module, so our /etc/modules.conf file contains the line:

```
alias eth0 qeth
```

Recommendations

Linux guests connected to a z/VM Guest LAN must communicate with the physical network through a z/VM TCP/IP or Linux router. This adds both latency and increased CPU utilization to the environment. It also means that it is impossible to participate in an external VLAN.[5] Additional subnetting is required in this environment, because the Linux guests must be on a separate subnet.

z/VM Guest LANs might, however, be appropriate in environments where physical network cards are limited, where there is a requirement for multiple Linux guests in the same z/VM LPAR to communicate with one another, and when the network activity of the Linux guests needs to be isolated from the physical network.

Given that VSWITCH can also fulfill these functions and does not have any of the drawbacks previously listed, it might be a more appropriate solution for your environment. We recommend that you use VSWITCH running in Layer 2 Switching mode rather than the standard z/VM Guest LAN.

[4] If you have a requirement to use DHCP for your Linux guests, we recommend that you implement the Layer 2 Switch version of VSWITCH; see “Layer 2 LAN Switching” on page 53 for details.
[5] We discuss VLANs in “VSWITCH” on page 44.

VSWITCH

The z/VM Virtual Switch (VSWITCH) introduced with z/VM V4.4 builds on the Guest LAN technology that was delivered in earlier z/VM releases. VSWITCH connects a Guest LAN to an external network using an OSA-Express interface. Two additional OSA-Express devices can be specified as backups in the VSWITCH definition. The Linux guests connected to the VSWITCH are on the same subnet as the OSA-Express interface or interfaces and other machines connected to that physical LAN segment.

The z/VM V4.4 implementation of VSWITCH operates at Layer 3 (network layer) of the OSI model. It only supports the transport of IP packets. In other words, it only can be used for TCP/IP applications. All destinations are identified as IP addresses, thus no MAC addresses are used (link layer independent), and ARP processing is offloaded to the OSA-Express adapter. In this environment, all hosts share the same OSA-Express MAC address. In a similar fashion to the description in “Address Resolution Protocol and OSA-Express” on page 12, all traffic destined for the physical portion of the LAN segment is encapsulated into an Ethernet frame with the OSA-Express’s MAC as the source MAC address. For inbound packets, the OSA-Express strips the Ethernet frame and forwards the IP packet to the Virtual Switch for delivery to the guest by the destination IP address within the IP packet.

In z/VM V5.1, the VSWITCH can operate at Layer 2 (data link layer) of the OSI model. We review this function in “Layer 2 LAN Switching” on page 53.

Because the VSWITCH is essentially connected to the physical LAN, the requirement for an intermediate router between the physical and (internal) Guest LAN segments is removed. This reduces network latency. It also reduces overall CPU consumption, in some test cases by as much 30%. Removing the router also means that you no longer need specialized skills to configure and administer a VM-based or Linux-based router.

Virtual LANs (VLANs) facilitate easy administration of logical groups of machines that can communicate as though they were on the same local area network (LAN). VSWITCH provides support for IEEE 802.1Q VLANs. This means that Linux guests connected to a VSWITCH can participate in a VLAN environment. Implementing VLAN in a z/VM V4.4 VSWITCH environment is covered in detail in ITSO Redpaper *Linux on IBM @server zSeries and S/390: VSWITCH and VLAN Features of z/VM 4.4*, REDP-3719.

VSWITCH controller

The VSWITCH’s connection to an OSA-Express interface is provided by a controller virtual machine. A controller is a z/VM service machine running the TCP/IP stack. At least one TCP/IP service machine must be configured to be a controller.

Note: Unlike previous networking configurations that used the z/VM TCP/IP stack, there is no requirement to manually configure IP addresses or devices when using VSWITCH. The only settings required in the stack’s PROFILE TCPIP file are listed in Figure 40 on page 48.

VSWITCH configuration

Complete the following required steps to implement VSWITCH:

1. Define a VSWITCH to act as a LAN segment for the virtual machines.
2. Configure one or more z/VM TCP/IP virtual machines to act as controllers for the VSWITCH.

3. Create a simulated Network Interface Card (NIC) on each virtual machine.
4. Link the Linux system to the VSWITCH.

Figure 37 shows a diagram of our test VSWITCH configuration.

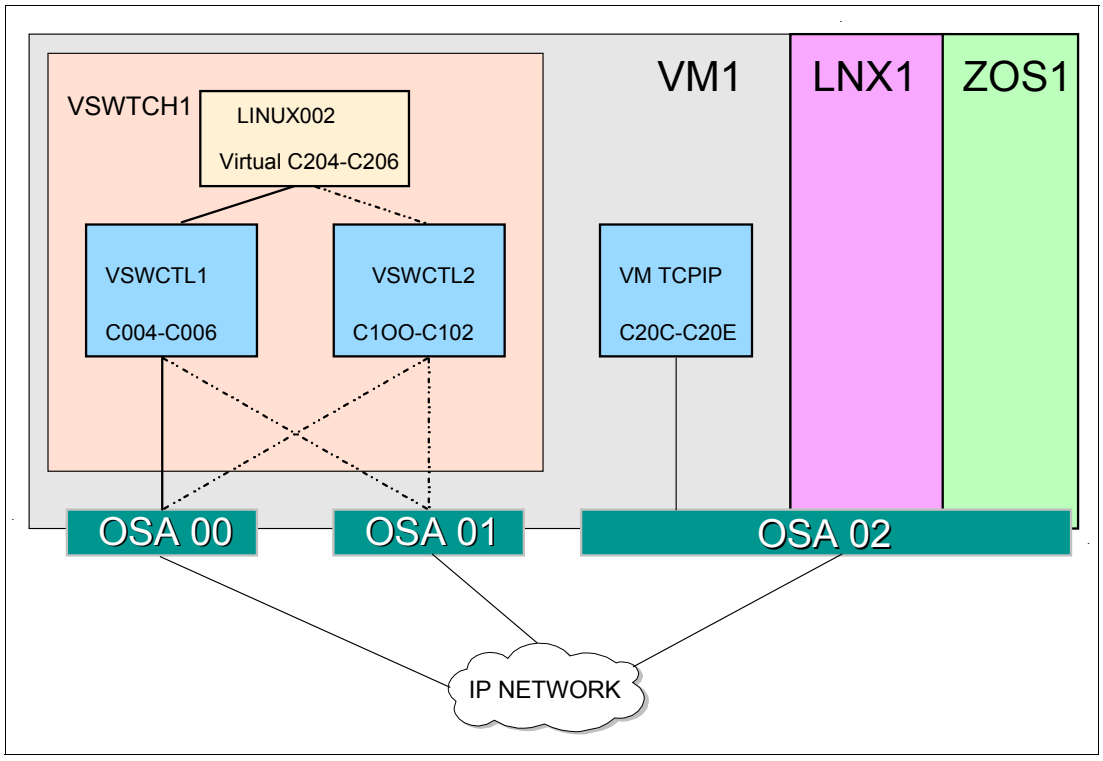

Figure 37 VSWITCH configuration

Important: The controller virtual machines (VSWCTL1 and VSWCTL2) are involved only in device initialization. Once initialized, data transfers occur directly between the OSA device and the Linux guest without passing through the controller stack.

Define the VSWITCH

A VSWITCH is created using the CP DEFINE VSWITCH command from a z/VM Class B user ID. Edit SYSTEM CONFIG, and add the following lines at the bottom of the file:

```
DEFINE VSWITCH VSWTCH1 RDEV C004 C100 PORTNAME OSACHP00 OSACHP01
MODIFY VSWITCH VSWTCH1 GRA LINUX002
```

This will create a VSWITCH called VSWTCH1 (see Figure 37). It will use two OSA devices, C004 to C006 (which is OSA 00) and C100 to C102 (OSA 01). We also granted Linux guest “LINUX002” access to the VSWITCH.

To do the same thing dynamically, we would have coded the following:

```
DEFINE VSWITCH VSWTCH1 RDEV C004 C100 PORTNAME OSACHP00 OSACHP01
SET VSWITCH VSWTCH1 GRANT LINUX002
```

Important: Any definitions created dynamically will not persist across IPLs.

The syntax of the DEFINE VSWITCH statement is as follows:

```
DEFINE VSWITCH switchname [ operands ]
```

Where:

switchname Is the name of the Virtual Switch.

operands Define the attributes of the Virtual Switch.

Table 11 summarizes the common operands accepted by the DEFINE VSWITCH command.

Table 11 Common operands of the DEFINE VSWITCH statement

LIMIT operands	Description
RDEV	A real device address to be used to connect the Virtual Switch to a QDIO OSA-Express device. You can specify a maximum of three real device numbers. Each real device address represents a trio of devices. For example, specifying RDEV 111 222 333 means that the first devices, 111-113, are used to provide the connection to the real hardware LAN segment. If there is a problem with the connection, devices 222-224 are used next to provide the connection, and if those devices fail to connect, devices 333-335 are used. This feature provides dynamic recovery for OSA-Express device failures.
CONnect	Indicates that the device identified by the RDEV keyword must be activated, and traffic must flow through the device to the real LAN segment.
CONTRoller * | *userid1*	Identifies the z/VM user ID that controls the OSA-Express device connected at the device address identified by rdev. CONTROLLER * means that CP selects from any of the eligible z/VM TCP/IP stacks. If you specify multiple real devices on the RDEV keyword, specify CONTROLLER *, or allow it to default. The controller functions are then spread across multiple z/VM TCP/IP stacks, providing more flexibility in case of a failure.
IP | Ethernet	Indicates whether the transport for the Virtual Switch is ETHERNET or IP. An ETHERNET Virtual Switch operates at the Layer 2[a] level of the OSI model, and an IP Virtual Switch operates at Layer 3.
PORTname *portname*	A 1-to-8 character name that identifies the OSA-Express adapter. You can specify a maximum of three port names. Multiple port names are used when different port names are needed for the multiple rdevs specified on the RDEV operand.

a. For a discussion about the Layer 2 Switch mode of VSWITCH, see “Layer 2 LAN Switching” on page 53.

Configure controller service machines

VSWITCH uses a TCP/IP service machine to initialize to an OSA-Express network interface. This TCP/IP service machine acts as a controller for the VSWITCH and manages the operation of the OSA-Express adapter ports the VSWITCH uses. In order for a VSWITCH to provide connectivity to a LAN, at least one TCP/IP service machine must be configured to be a controller. The configuration allows the TCP/IP stack to connect to the system service that manages VSWITCH connections. The TCP/IP service machine is then able to act as a controller for a VSWITCH OSA connection. In our example, we defined two virtual machines to act as VSWITCH controllers.

Edit the User Directory and add the lines shown in Figure 38 on page 47.

```
USER VSWCTL1 VSWCTL1 32M 128M ABG
INCLUDE TCPCMSU
OPTION QUICKDSP SVMSTAT MAXCONN 1024 DIAG98
SHARE RELATIVE 3000
IUCV *VSWITCH MSGLIMIT 65535
LINK 4TCPIP40 491 491 RR
LINK 4TCPIP40 492 492 RR
LINK TCPMAINT 591 591 RR
LINK TCPMAINT 592 592 RR
LINK TCPMAINT 198 198 RR
MDISK 191 3390 1706 005 440W02 MR RTCPIP WTCPIP MTCPIP

USER VSWCTL2 VSWCTL2 32M 128M ABG
INCLUDE TCPCMSU
OPTION QUICKDSP SVMSTAT MAXCONN 1024 DIAG98
SHARE RELATIVE 3000
IUCV *VSWITCH MSGLIMIT 65535
LINK 4TCPIP40 491 491 RR
LINK 4TCPIP40 492 492 RR
LINK TCPMAINT 591 591 RR
LINK TCPMAINT 592 592 RR
LINK TCPMAINT 198 198 RR
MDISK 191 3390 1712 005 440W02 MR RTCPIP WTCPIP MTCPIP
```

Figure 38 VSWITCH controller user definitions

Note that the user entries include a IUCV *VSWITCH statement, which is required for the machines to be considered as VSWITCH controllers.

AUTOLOG changes

In order for the controller service machines to start automatically when VM is IPLed, we need to add some entries in the AUTOLOG1 PROFILE EXEC, as shown in Figure 39. Although we added MODIFY VSWITCH statements to grant access to the VSWITCH for specific Linux guests into our SYSTEM CONFIG file, the example in Figure 39 shows how you would use the SET VSWITCH command to do the same thing using AUTOLOG.

```
ADDRESS COMMAND CP XAUTOLOG VSWCTL1
ADDRESS COMMAND CP XAUTOLOG VSWCTL2
ADDRESS COMMAND CP SET VSWITCH VSWTCH1 GRANT LINUX001
ADDRESS COMMAND CP SET VSWITCH VSWTCH1 GRANT LINUX002
ADDRESS COMMAND CP SET VSWITCH VSWTCH1 GRANT LINUX003
```

Figure 39 PROFILE EXEC for AUTOLOG1

PROFILE TCPIP and PROFILE EXEC

Copy an existing z/VM TCP/IP stack's PROFILE TCPIP file to the 191 disk of each controller service machine. You can usually find an existing copy on the TCPMAINT 198 disk. However, you only need to include the lines shown in Figure 40 on page 48 (so it is probably easier to create one).

```
OBEY
OPERATOR MAINT
ENDOBEY
VSWITCH CONTROLLER ON
```

Figure 40 VSWITCH controller service machine's PROFILE TCPIP

Copy an existing VM TCP/IP service machine's PROFILE EXEC file to the 191 disk of each controller service machine.

The SYSTEM DTCPARMS file

Create a SYSTEM DTCPARMS file on each controller's 191 disk. You can usually find an existing copy on the TCPMAINT 198 disk. However, you only need to include the lines shown in Figure 41.

```
:NICK.VSWCTL1 :TYPE.SERVER
:class.stack
:NICK.VSWCTL2 :TYPE.SERVER
:class.stack
```

Figure 41 The SYSTEM DCTPARMS file

Linking the Linux guest to the VSWITCH

Make sure that the Linux guest has a NICDEF entry for its simulated NIC in the User Directory:

```
NICDEF  C204 TYPE QDIO LAN SYSTEM VSWTCH1
```

Linux definitions

From a Linux perspective, make sure that the /etc/chandev.conf file has an entry as follows:

```
noauto;qeth0,0xc204,0xc205,0xc206;add_parms,0x10,0xc204,0xc206,portname:OSACHP01
```

Note: Although we use a port name of OSACHP01 and have devices defined that do not match the actual OSA devices, in fact, we are simply connecting to the VSWITCH (by way of the NICDEF statement). It determines which devices and OSA port names are really going to be used.

Make sure that you have a valid /etc/sysconfig/network/ifcfg-eth0 file. We gave this Linux guest a hard-coded IP address, as shown in Figure 42.

```
BOOTPROTO="static"
STARTMODE="onboot"
IPADDR="9.190.207.96"
MTU="1500"
NETMASK="255.255.255.0"
NETWORK="255.255.255.0"
BROADCAST="255.255.255.0"
```

Figure 42 The ifcfg-eth0 file

Note: Because we are running a test system, we now re-IPL z/VM to introduce the changes. We could have made all the changes dynamically; this might be something you need to do if you are not running in a test system where you can re-IPL z/VM whenever you need to.

Verifying the configuration

Now check to see if the changes that have been made are correct. Note that the output from the z/VM QUERY commands is from a z/VM V4.4 system. The output from these commands has changed in z/VM V5.1.

First, we check to see if the VM TCP/IP stacks have been recognized as controller service machines using the QUERY CONTROLLER, as command shown in Figure 43.

```
QUERY CONTROLLER
CONTROLLER VSWCTL1   Available: YES   VDEV Range: *
  SYSTEM VSWTCH1     Controller: *
CONTROLLER VSWCTL2   Available: YES   VDEV Range: *
```

Figure 43 The QUERY CONTROLLER command output

Next, we check the VSWITCH itself using the QUERY VSWITCH command, as shown in Figure 44.

```
QUERY VSWITCH
VSWITCH SYSTEM VSWTCH1  Type: VSWITCH  Active: 0    MAXCONN: INFINITE
  PERSISTENT  RESTRICTED    NONROUTER  MFS: 8192     ACCOUNTING: OFF
  State: Ready
  CONTROLLER: VSWCTL1       IPTIMEOUT: 5             QUEUESTORAGE: 8
  PORTNAME: OSACHP00        RDEV: C004 VDEV: C004
  PORTNAME: OSACHP01        RDEV: C100
```

Figure 44 The QUERY VSWITCH command output

Now, IPL your Linux guest (LINUX002 in our example). Verify that it has connectivity by pinging its default gateway. You can also use the QUERY VSWITCH DETAILS command (shown in Figure 45 on page 50) to determine if the Linux system is now part of the VSWITCH Guest LAN.

```
Q VSWITCH DETAILS
VSWITCH SYSTEM VSWTCH1  Type: VSWITCH  Active: 1 MAXCONN: INFINITE
  PERSISTENT  RESTRICTED   NONROUTER  MFS: 8192    ACCOUNTING: OFF
  State: Ready
  CONTROLLER: VSWCTL1      IPTIMEOUT: 5          QUEUESTORAGE: 8
  PORTNAME: OSACHP00       RDEV: C004 VDEV: C004
  PORTNAME: OSACHP01       RDEV: C100
  RX Packets: 520 Discarded: 0         Errors: 0
  TX Packets: 280 Discarded: 0         Errors: 0
  RX Bytes: 1245 TX Bytes: 4312
    Authorized userids:
      LINUX001 VLAN:  ANY
      LINUX002 VLAN:  ANY
      LINUX003 VLAN:  ANY
     SYSTEM   VLAN:  ANY
    VSWITCH Connection:
      Device: C006  Unit: 002   Role: DATA
      VLAN: ANY  Assigned by user
        Unicast IP Addresses:
          9.190.207.1     Mask: 0.0.0.0        Remote
Adapter Owner: LINUX002 NIC: C204  Name: OSACHP02
       Device: C206  Unit: 002   Role: DATA
       VLAN: ANY  Assigned by user
         Unicast IP Addresses:
           9.190.207.96       Mask: 255.255.255.0
           FE80::200:0:100:4/64                    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:FF00:4   MAC: 33-33-FF-00-00-04 Local
```

Figure 45 The QUERY VSWITCH DETAILS command output

z/VM V4.4 VSWITCH failover support

Failover support for Virtual Switches provides recovery for controller failures and OSA-Express card failures. In our example configuration, we use two controllers and two separate OSA cards. If one controller fails, the other controller will take over. Likewise, if one OSA fails, traffic will be switched to use the second OSA. Using Figure 37 on page 45 as an example, controller VSWCTL1 initializes OSA00 to form the active path between the external network and the VSWTCH1 Guest LAN. Controller VSWCTL2 is available as a backup controller, and OSA01 is available as a backup OSA-Express device if needed.

In the event of an OSA failure, the primary device addresses are detached from the controller TCP/IP stack. The IP addresses are deleted from the failing OSA-Express card. The backup devices are attached to an available controller service machine. This controller then initializes and starts the backup device. The IP addresses are loaded into the backup OSA-Express device, and any lost datagrams must then be retransmitted.

z/VM V4.4 VSWITCH testing failover support

We tested failover by removing OSA CHPIDs (through the zSeries Hardware Management Console) and using the CP FORCE command to stop controllers. While doing this, we had a Linux guest pinging a machine on the external network. Figure 46 on page 51 shows the sequence.

```
BEFORE WE CONFIGURED OSA 00 OFFLINE

QUERY VSWITCH
VSWITCH SYSTEM VSWTCH1  Type: VSWITCH  Active: 3    MAXCONN: INFINITE
  PERSISTENT  RESTRICTED   NONROUTER  MFS: 8192    ACCOUNTING: OFF
  State: Ready
  CONTROLLER: VSWCTL1      IPTIMEOUT: 5           QUEUESTORAGE: 8
  PORTNAME: OSACHP00       RDEV: C004 VDEV: C004
   PORTNAME: OSACHP01        RDEV: C100

Q c100-c102
OSA  C100 FREE    , OSA  C101 FREE    , OSA  C102 FREE


AFTER WE CONFIGURED OSA 00 OFFLINE

QUERY VSWITCH
VSWITCH SYSTEM VSWTCH1  Type: VSWITCH  Active: 3    MAXCONN: INFINITE
  PERSISTENT  RESTRICTED   NONROUTER  MFS: 8192    ACCOUNTING: OFF
  State: Ready
  CONTROLLER: VSWCTL1      IPTIMEOUT: 5           QUEUESTORAGE: 8
  PORTNAME: OSACHP00       RDEV: C004
   PORTNAME: OSACHP01        RDEV: C100 VDEV: C100

Q c100-c102
OSA  C100 ATTACHED TO VSWCTL1  C100
OSA  C101 ATTACHED TO VSWCTL1  C101
OSA  C102 ATTACHED TO VSWCTL1  C102
```

Figure 46 Simulated OSA card failure

Before we configured OSA 00 offline, we can see that the backup OSA device has not been preinitialized in any way. The backup devices (C100-C102) are online and free. After we remove OSA 00, the controller uses OSA 01 (devices C100-C102). We did not see any noticeable lag in the output from the **ping** command.

After we forced off the VSWCTL1 controller, the system switched to using VSWCTL2, as shown in Figure 47 on page 52. Again, we did not see any noticeable lag in the output from the **ping** command.

```
BEFORE FORCING OFF CONTROLLER VSWCTL1

q controller
CONTROLLER VSWCTL1   Available: YES   VDEV Range: *
  SYSTEM VSWTCH1     Controller: *
CONTROLLER VSWCTL2   Available: YES   VDEV Range: *

q vswitch
VSWITCH SYSTEM VSWTCH1  Type: VSWITCH  Active: 3    MAXCONN: INFINITE
  PERSISTENT  RESTRICTED   NONROUTER  MFS: 8192    ACCOUNTING: OFF
  State: Ready
  CONTROLLER: VSWCTL1      IPTIMEOUT: 5           QUEUESTORAGE: 8
  PORTNAME: OSACHP00       RDEV: C004
   PORTNAME: OSACHP01        RDEV: C100 VDEV: C100

force vswctl1
USER DSC   LOGOFF AS  VSWCTL1  USERS = 16    FORCED BY MAINT

AFTER FORCING OFF CONTROLLER VSWCTL1

q controller
CONTROLLER VSWCTL2   Available: YES   VDEV Range: *
  SYSTEM VSWTCH1     Controller: *
Ready; T=0.01/0.01 07:18:33
q vswitch
VSWITCH SYSTEM VSWTCH1  Type: VSWITCH  Active: 3    MAXCONN: INFINITE
  PERSISTENT  RESTRICTED   NONROUTER  MFS: 8192    ACCOUNTING: OFF
  State: Ready
  CONTROLLER: VSWCTL2      IPTIMEOUT: 5           QUEUESTORAGE: 8
   PORTNAME: OSACHP00        RDEV: C004 VDEV: C004
   PORTNAME: OSACHP01        RDEV: C100
```

Figure 47 Simulated controller service machine failure

Figure 48 shows the system messages when moving over to a backup controller.

```
07:18:29 HCPSWU2830I VSWITCH SYSTEM VSWTCH1 status is controller not available.
07:18:29 OSA   C100 DETACHED VSWCTL1  C100 BY VSWCTL1
07:18:29 OSA   C101 DETACHED VSWCTL1  C101 BY VSWCTL1
07:18:29 OSA   C102 DETACHED VSWCTL1  C102 BY VSWCTL1
07:18:29 HCPSWU2830I VSWITCH SYSTEM VSWTCH1 status is ready.
07:18:29 HCPSWU2830I VSWCTL2 is VSWITCH controller.
```

Figure 48 System messages

z/VM V5.1 VSWITCH failover support

VSWITCH failover has been enhanced in z/VM V5.1. The backup controller is preinitialized, and the backup OSA-Express devices are preattached to the backup controller (see Figure 49 on page 53). In the event of an OSA failure, the VSWITCH is placed into a new “RECOVERING” state. The primary device addresses are detached from the controller TCP/IP stack, and the IP addresses are deleted from the failing OSA. The IP addresses are then added, and data transfer queues are swapped to the preinitialized backup OSA-Express card.

```
q controller
Controller VSWCTL1   Available: YES   VDEV Range: *         Level 510
  Capability: IP ETHERNET ARP_VLAN
   SYSTEM VSWTCH1    Primary          Controller: *         VDEV: C004
Controller VSWCTL2   Available: YES   VDEV Range: *         Level 510
  Capability: IP ETHERNET ARP_VLAN
   SYSTEM VSWTCH1    Backup           Controller: *         VDEV: C100

q vswitch
VSWITCH SYSTEM VSWTCH1  Type: VSWITCH  Connected: 1    Maxconn: INFINITE
  PERSISTENT  RESTRICTED     ETHERNET                  Accounting: OFF
   VLAN Unaware
   State: Ready
   QueueStorage: 8
  Portname: OSACHP00   RDEV: C004 Controller: VSWCTL1  VDEV:  C004
  Portname: OSACHP01   RDEV: C100 Controller: VSWCTL1  VDEV:  C100 BACKUP
```

Figure 49 z/VM V5.1 enhanced VSWITCH failover

This means that recovery time is very fast, because a lot of the set-up work for recovery has been done in advance.

Recommendations

VSWITCH is a very powerful virtualization solution. It removes the requirement for an intermediate router between an external LAN and the internal Guest LAN. This can save significant amounts of CPU cycles if your environment is moderately to heavily used. VSWITCH supports the IEEE 802.1Q VLAN standards, which are important to many organizations. The failover capabilities of VSWITCH are very impressive. With only a minimal amount of effort, it is possible to configure a highly available networking environment.

As previously outlined, VSWITCH provides significant benefits over the older point-to-point technologies and has more function than the base Guest LAN implementation. For communications within a single z/VM environment, VSWITCH is a very good solution. An even better solution, however, is VSWITCH running in Layer 2 Switch mode, which is the topic of the next section.

Layer 2 LAN Switching

You will recall that in z/VM V4.4, the VSWITCH operates at Layer 3 (network layer) of the OSI model. It only supports the transport of IPv4 packets and so is limited to supporting only applications that use the TCP/IP protocol. All destinations are identified as IP addresses, thus no MAC addresses are used (data link layer independent), and ARP processing is offloaded to the OSA-Express adapter. In this environment, all hosts share the same OSA-Express MAC address.

In z/VM V5.1, VSWITCH has been enhanced to operate in one of two modes. In IP mode, the VSWITCH operates at Layer 3 (network layer) of the OSI model just as it did in z/VM V4.4.

In Ethernet mode, VSWITCH operates at Layer 2 (data link layer) of the OSI model. When operating in Ethernet mode, each guest has a unique MAC address that VSWITCH uses to forward frames. Data is transported and delivered within Ethernet frames, providing the ability to transport both IP and non-IP application data through the fabric that the Virtual Switch supports.

Through the Address Resolution Protocol (ARP) processing of each guest, the guest's MAC address is stored in the ARP cache of hosts residing on the physical side of the LAN segment. Generation and assignment of the locally defined MAC address is performed by z/VM under the control of the LAN administrator. Each outbound or inbound frame through the OSA-Express switch trunk connection is an Ethernet frame with the guest's MAC address as the source or destination MAC address.

The ability to support IPv6 and also non-TCP/IP protocols, such as SNA, DECnet, IPX, or NetBIOS, means that it is now possible to consolidate systems that were previously unable to be considered for workload consolidation.

Restriction: As discussed in "LPAR-to-LPAR communication" on page 10, traffic can be sent between LPARs without going over the physical network when the port sharing capability of the OSA-Express card is used. A restriction exists with port sharing now that Layer 2 mode has been introduced.

Port sharing is only supported between Virtual Switches that are of the same transport mode, for example Layer 2 with Layer 2 and Layer 3 with Layer 3. Attempted communications between a Layer 2 Virtual Switch and a Layer 3 Virtual Switch sharing the same OSA-Express adapter will result in a network timeout condition. To resolve this, you should have the Layer 2 Virtual Switch and the Layer 3 Virtual Switch on separate OSA-Express adapters that are connected to the same LAN segment. With this solution, the communication between these Virtual Switches is now sent out onto the physical LAN segment, and full MAC resolution will be achieved.

Requirements

Restriction: Layer 2 Switch support is only available for the zSeries 990 and 890 servers at Driver Level 55K.

In order to use Layer 2 LAN Switching, you need the following prerequisites:

- OSA-Express code level 6.23 for z990 and z890 (available in October 2004).
- z/VM V5.1 with APAR VM63538.
- Linux qeth device driver that supports Layer 2. Layer 2 support for the "June 2003 stream" (kernel 2.4.21) was released on developerWorks in October 2004. Check with your distributor to see if Layer 2 support is available in your Linux distribution. Layer 2 support for kernel Version 2.6 ("April 2004 stream") is not available at this time. In our testing, we used SLES8 running at kernel 2.4.21-231 with a privately built qeth driver.

Important: To determine if you have the correct level of the qeth device driver, check the revision number. If the revision number is lower than 1.397, Layer 2 support will not work. To check the revision number from Linux:

```
strings /lib/modules/2.4.21-231-default/kernel/drivers/s390/net/qeth.o | grep driver
```

The response back will look similar to the following:

```
qeth S/390 OSA-Express driver ($Re -L2- : 1.397 $/$Re -L2- : 1.131 $/$Re -L2- : 1.50
$:IPv6:VLAN)
```

MAC address generation

The LAN administrator decides what MAC addresses are locally generated and associated with each guest. They do this using a combination of the VMLAN statement (in SYSTEM CONFIG) and the NICDEF statement (in the User Directory).

The VMLAN statement contains a MACPREFIX parameter that enables the administrator to specify a 3-byte manufacturer ID prefix for all MAC addresses in this z/VM system. The VMLAN parameter MACIDRANGE controls the range of identifiers that can be used by CP when generating the unique identifier component of a virtual NIC's MAC address.

In the User Directory, a NICDEF statement is added for each guest that will be connected to the VSWITCH. The MACID parameter of NICDEF enables the administrator to specify a unique identifier that will be appended to the MACPREFIX to form a unique MAC address for that guest. If MACID is omitted, the CP will generate a unique identifier based on the range specified in the VMLAN MACIDRANGE parameter. If you specify your own MACID value in the NICDEF, and the MACID is already in use, the adapter will not be created, and you will need to remove the conflicting device or select a different MACID.

These locally generated MAC addresses will be visible across the physical portion of the LAN segment.

Tip: If you run multiple z/VM systems on the same CEC, you should change the MACPREFIX of each system to avoid MAC address duplication.

Configuring a Layer 2 Switch

The following steps are required to implement a Layer 2 Switch. They are very similar to the steps we followed to build an IP mode VSWITCH (see "VSWITCH configuration" on page 44).

1. Define a VSWITCH to act as a LAN segment for the virtual machines.
2. Configure one or more z/VM TCP/IP virtual machines to act as controllers for the VSWITCH.
3. Create a simulated Network Interface Card (NIC) on each virtual machine.
4. Link the Linux guest to the VSWITCH.

Figure 50 on page 56 shows a diagram of our test Layer 2 VSWITCH configuration.

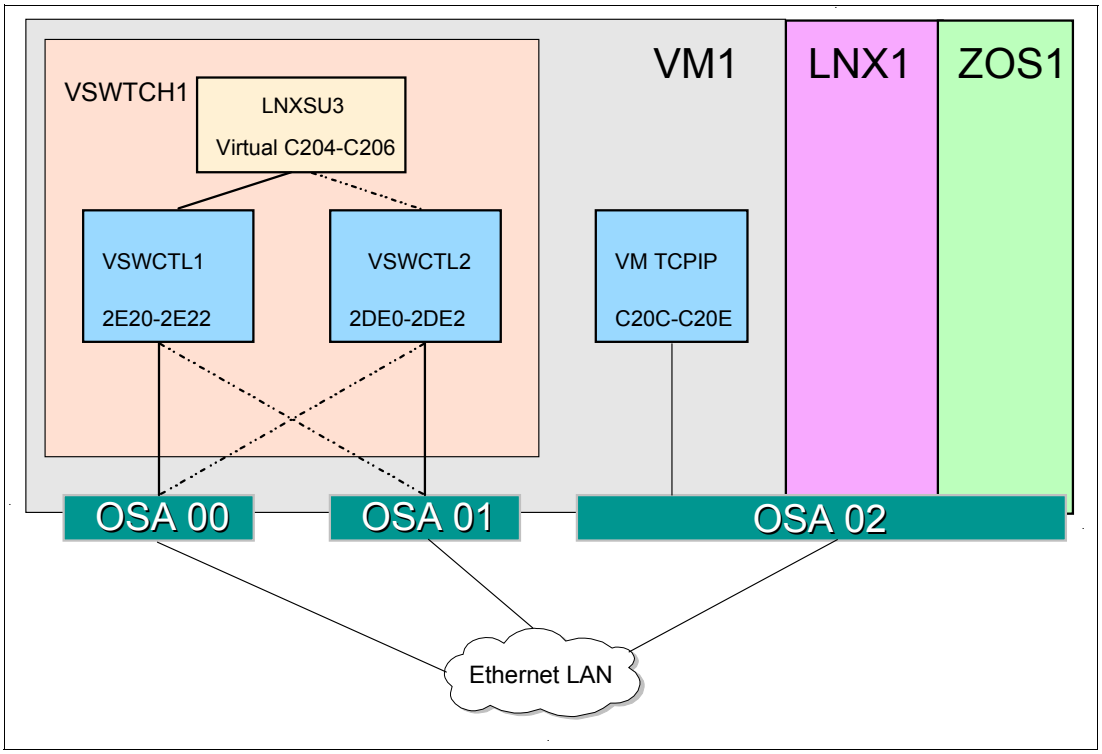

Figure 50 Layer 2 VSWITCH configuration

Important: The controller virtual machines (VSWCTL1 and VSWCTL2) are involved only in device initialization. Once initialized, data transfers occur directly between the OSA device and the Linux guest without passing through the controller stack.

Define the VSWITCH

A VSWITCH is created using the CP DEFINE VSWITCH command from a z/VM Class B user ID.

Edit SYSTEM CONFIG, and add the following lines at the bottom of the file:

```
DEFINE VSWITCH VSWTCH1 RDEV 2E20 2DE0 CON CONTR * ETHERNET PORTNAME OSA2E20 OSA2DE0
MODIFY VSWITCH VSWTCH1 GRA LNXSU3
```

The ETHERNET parameter means that this VSWITCH will operate in Layer 2 Switch mode. The VSWITCH will be connected to two OSA-Express cards, devices 2E20 to 2E22 (OSA 00) and 2DE0 to 2DE2 (OSA 01). We also granted Linux guest “LNXSU3” access to the VSWITCH.

The syntax of the DEFINE VSWITCH statement is as follows:

```
DEFINE VSWITCH switchname [ operands ]
```

Where:

`switchname` Is the name of the Virtual Switch.

`operands` Define the attributes of the Virtual Switch.

Table 12 summarizes the common operands accepted by the DEFINE VSWITCH command.

Table 12 Common operands of the DEFINE VSWITCH statement

LIMIT parameter operands	Description
RDEV	A real device address to be used to connect the Virtual Switch to a QDIO OSA-Express device. You can specify a maximum of three real device numbers. Each real device address represents a trio of devices. For example, specifying RDEV 111 222 333 means that the first devices, 111-113, are used to provide the connection to the real hardware LAN segment. If there is a problem with the connection, devices 222-224 are used next to provide the connection, and if those devices fail to connect, devices 333-335 are used. This feature provides dynamic recovery for OSA-Express device failures.
CONnect	Indicates that the device identified by the RDEV keyword must be activated, and traffic must flow through the device to the real LAN segment.
CONTRoller * | *userid1*	Identifies the z/VM user ID that controls the OSA-Express device connected at the device address identified by rdev. CONTROLLER * means that CP selects from any of the eligible z/VM TCP/IP stacks. Specify CONTROLLER *, or allow it to default. The controller functions are then spread across multiple z/VM TCP/IP stacks, providing more flexibility in case of a failure.
IP | Ethernet	Indicates whether the transport for the Virtual Switch is ETHERNET or IP. An ETHERNET Virtual Switch operates at the Layer 2 level of the OSI model, and an IP Virtual Switch operates at Layer 3.
PORTname *portname*	A 1-to-8 character name that identifies the OSA-Express adapter. You can specify a maximum of three port names. Multiple port names are used when different port names are needed for the multiple rdevs specified on the RDEV operand.

The VMLAN statement

In addition to the DEFINE LAN statement, we also add a VMLAN statement to the CP SYSTEM CONFIG file, because we want to establish system-wide MAC address definitions that will be used when generating local MAC addresses for individual guests.

The syntax of the VMLAN statement is as follows:

```
VMLAN LIMIT [ operands ]
```

Where *operands* define the attributes to be set for all z/VM Guest LANs in the system.

Table 13 summarizes the operands of the VMLAN command that we want to use.

Table 13 Operands of the VMLAN statement

LIMIT parameter operands	Description
MACPREFIX *macprefix*	Specifies the 3-byte prefix (manufacturer ID) used when generating locally administered MAC addresses on the system. It must be six hexadecimal digits within the range of 020000 through 02FFFF (inclusive). In combination with the MAC ID used on the NICDEF directory statement, the MACPREFIX allows unique identification of virtual adapters within a network. If MACPREFIX is not specified, the default is 020000 (02-00-00).

LIMIT parameter operands	Description
MACIDRange SYSTEM *xxxxxx-xxxxxx* USER *xxxxxx-xxxxxx*	
MACIDRange SYSTEM *xxxxxx-xxxxxx*	The range of identifiers (up to six hexadecimal digits each) to be used by CP when generating the unique identifier part (last six hexadecimal digits) of a virtual adapter MAC address. If a SYSTEM MACIDRANGE is not specified, CP creates unique identifiers in any range (000001-FFFFFF).
USER *xxxxxx-xxxxxx*	The subset of the SYSTEM range of identifiers that are reserved for user definition of MACIDs in the NICDEF directory statement. When specified, CP does not assign MACIDs within this USER range during creation of virtual adapters defined dynamically (DEFINE NIC) or with the NICDEF (or SPECIAL) directory statement without the MACID operand. In these cases, CP generates a unique identifier for the adapter outside of the USER range. Any MACID values specified on a NICDEF directory statement must be within the USER range or the virtual adapter is not defined during LOGON processing. If a USER MACIDRANGE is not specified, CP creates unique identifiers within the SYSTEM MACIDRANGE.

On our test system, we used the following VMLAN definitions in the SYSTEM CONFIG file, as shown in Figure 51.

```
VMLAN MACPREFIX 02EEEE
VMLAN MACIDRANGE SYSTEM 100000-1FFFFF
```

Figure 51 VMLAN definition

Configure controller service machines

The configuration of the VSWITCH controller service machines for a Layer 2 VSWITCH is exactly the same as for the “original” Layer 3 VSWITCH. “Configure controller service machines” on page 46 provides an example of how to set up these service machines.

Verifying the configuration

We then re-IPLed our test system and checked to ensure that the changes we made were correct. First, we checked to see if the VM TCP/IP stacks have been recognized as controller service machines, as shown in Figure 52.

```
QUERY CONTROLLER
Controller VSWCTL1   Available: YES   VDEV Range: *         Level 510
  Capability: IP ETHERNET ARP_VLAN
   SYSTEM VSWTCH1    Primary           Controller: *         VDEV: 2E20
Controller VSWCTL2   Available: YES   VDEV Range: *         Level 510
  Capability: IP ETHERNET ARP_VLAN
   SYSTEM VSWTCH1    Backup            Controller: *         VDEV: 2DE0
```

Figure 52 QUERY CONTROLLER output

Next, we checked the VSWITCH itself, as shown in Figure 53 on page 59.

```
QUERY VSWITCH
VSWITCH SYSTEM VSWTCH1  Type: VSWITCH Connected: 0    Maxconn: INFINITE
  PERSISTENT  RESTRICTED    ETHERNET                  Accounting: OFF
  VLAN Unaware
  State: Ready
  QueueStorage: 8
  Portname: OSA2E20    RDEV: 2E20 Controller: VSWCTL1  VDEV:  2E20
  Portname: OSA2       RDEV: 2DE0 Controller: VSWCTL2  VDEV:  2DE0 BACKUP
```

Figure 53 QUERY VSWITCH output

Finally, we checked to see if the VMLAN changes have been applied, as shown in Figure 54.

```
Q VMLAN
VMLAN maintenance level:
  Latest Service: Base
VMLAN MAC address assignment:
  MACADDR Prefix: 02EEEE
  MACIDRANGE SYSTEM: 100000-1FFFFF
             USER:   000000-000000
VMLAN default accounting status:
  SYSTEM Accounting: OFF       USER Accounting: OFF
VMLAN general activity:
  PERSISTENT Limit: INFINITE   Current: 1
  TRANSIENT  Limit: INFINITE   Current: 0
```

Figure 54 QUERY VMLAN output

Linking the Linux guest to the VSWITCH

Make sure the Linux guest has a NICDEF entry for its simulated NIC in the User Directory:

```
NICDEF  C204 TYPE QDIO LAN SYSTEM VSWTCH1
```

Note: We could have added a MACID parameter to the NICDEF statement. Instead, we chose to let the system generate one for us.

After you have logged on to the Linux system, check the network interface as follows:

```
QUERY NIC
Adapter C204  Type: QDIO      Name: UNASSIGNED  Devices: 3
  Port 0 MAC: 02-EE-EE-10-00-00  VSWITCH: SYSTEM VSWTCH1
```

We chose 02EEEE as our MACPREFIX and 100000-1FFFFF as our MACIDRANGE using VMLAN statements. Note that the MAC address is 02-EE-EE-10-00-00, which is the first MAC address to be allocated in that range.

Testing the Layer 2 Switch

In order to test our environment, we connected a Linux guest to the Layer 2 Virtual Switch. Using a DHCP client, the guest requested an IP address from a DHCP server on an external LAN segment. The DHCP client running on the Linux guest broadcasts a DHCPDISCOVER message that includes its MAC address. This message is used to request an IP address from any DHCP server that is listening. A DHCP server will send back a DHCPOFFER message to the guest. The DHCPOFFER gets to its destination, because the Linux guest has a valid MAC address that is visible both on the internal and external (physical) parts of the LAN segment.

Our Linux guest used a /etc/chandev.conf entry as follows:

```
noauto;qeth0,0xc204,0xc205,0xc206;add_parms,0x10,0xc204,0xc206,layer2,portname:OSA2E20
```

Note: When using the Layer 2 Switch, you need to use a new qeth chandev parameter, `layer2`, that enables Layer 2 functionality. There is also a parameter called `no_layer2` that disables Layer 2 functionality. You could use this if you are connecting to a Layer 3 (IP) VSWITCH. If you do not use either of the parameters, the qeth driver will expect to connect to a Layer 3 (IP) VSWITCH.

Make sure you have a valid /etc/sysconfig/network/ifcfg-eth0 file. In Figure 55, the Linux guest uses a DHCP client to lease an IP address.

```
BOOTPROTO='dhcp'
STARTMODE='onboot'
IPADDR=
MTU='1500'
NETMASK=
NETWORK=
BROADCAST=
DATA_CHANNEL=''
PORTNAME=''
READ_CHANNEL=''
REMOTE_IPADDR=''
UNIQUE=''
WIRELESS='no'
WRITE_CHANNEL=''
```

Figure 55 The ifcfg-eth0 file

Reboot the Linux guest after you have made these changes. We now use the **ifconfig** command in Figure 56 to determine if the lease of an IP address was successful.

```
ifconfig
eth0      Link encap:Ethernet  HWaddr 02:EE:EE:10:00:00
          inet addr:9.12.4.160  Bcast:9.12.5.255  Mask:255.255.254.0
          inet6 addr: fe80::2ee:ee00:10:0/64 Scope:Link
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:79 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:4946 (4.8 Kb)  TX bytes:1066 (1.0 Kb)
          Interrupt:7
```

Figure 56 ifconfig output

We have now successfully leased an IP address (9.12.4.160) from an external DHCP server. We can also verify this from a VSWITCH perspective by using a QUERY VSWITCH command, as shown in Figure 57 on page 61.

```
QUERY VSWITCH DETAILS
VSWITCH SYSTEM VSWTCH1  Type: VSWITCH Connected: 1   Maxconn: INFINITE
 PERSISTENT  RESTRICTED   ETHERNET             Accounting: OFF
 VLAN Unaware
 State: Ready
 QueueStorage: 8
 Portname: OSA2E20   RDEV: 2E20 Controller: VSWCTL1  VDEV:  2E20
 Portname: OSA2     RDEV: 2DE0 Controller: VSWCTL2  VDEV:  2DE0 BACKUP
   VSWITCH Connection:
    RX Packets: 832      Discarded: 0        Errors: 0
    TX Packets: 324      Discarded: 0        Errors: 0
    RX Bytes: 62854            TX Bytes: 2780423
    Device: 2E22  Unit: 002   Role: DATA
      Unicast IP Addresses:
       9.12.4.92         MAC: 00-07-85-85-65-C2 Remote
       9.12.4.156        MAC: 00-02-55-E4-62-0E Remote
  Adapter Owner: LNXSU3   NIC: C204  Name: OSA2E20
    RX Packets: 832      Discarded: 0        Errors: 0
    TX Packets: 363      Discarded: 0        Errors: 0
    RX Bytes: 62854            TX Bytes: 43378
    Device: C206  Unit: 002   Role: DATA
    Options: Ethernet Broadcast
      Unicast MAC Addresses:
       02-EE-EE-10-00-00 IP: 9.12.4.160
       Multicast MAC Addresses:
         01-00-5E-00-00-01 IP: 224.0.0.1
         33-33-00-00-00-01 IP: FF02::1
         33-33-FF-10-00-00 IP: FF02::FF10:0
```

Figure 57 QUERY VSWITCH output

Recommendations

Layer 2 Switching support, now available as part of z/VM V5.1 VSWITCH, is a very powerful solution that builds on earlier z/VM LAN technology. As with the initial z/VM V4.4 implementation, this version of VSWITCH removes the requirement for an intermediate router between an external LAN and the internal Guest LAN. Running as a Layer 2 Switch means that Linux guests can exist and operate in the network in exactly the same way as physical machines. Non-IP protocols, such as SNA, IPX, NetBIOS, and DECnet, are now supported, as is IPv6, which means that there are more opportunities to consolidate machines to a z/VM environment.

We recommend that you run VSWITCH running in Layer 2 mode if you have a requirement to run multiple Linux systems in a z/VM environment. This option is the current “state of the art” network virtualization technology for z/VM.

Performance considerations

When choosing a networking option in a z/VM environment, it is important to understand the relative performance of the different options. From Figure 58 on page 62, we can draw a number conclusions:

- LCS devices are the least attractive option. They have a very low throughput and a high CPU overhead.

- The Gigabit Ethernet card (in particular, using a 1500 byte MTU size), while providing good bandwidth to the external network, has a fairly high CPU cost. It is also important to note that by default, the Linux qdio device driver reserves approximately 8 MB of storage *for each QDIO device*. This storage is locked in real storage frames below the 2 GB line. This cost needs to be considered when running multiple Linux guests (particularly if each guest uses multiple QDIO devices). For example, the requirement for 50 Linux guests sharing an OSA adapter equates to 400 MB of z/VM real storage.
- The z/VM Guest LANs (HiperSockets or QDIO) provide excellent bandwidth at very low CPU cost. For communications between guests within a z/VM LPAR, these are the most efficient options.
- HiperSockets is the best method for communicating between LPARs.

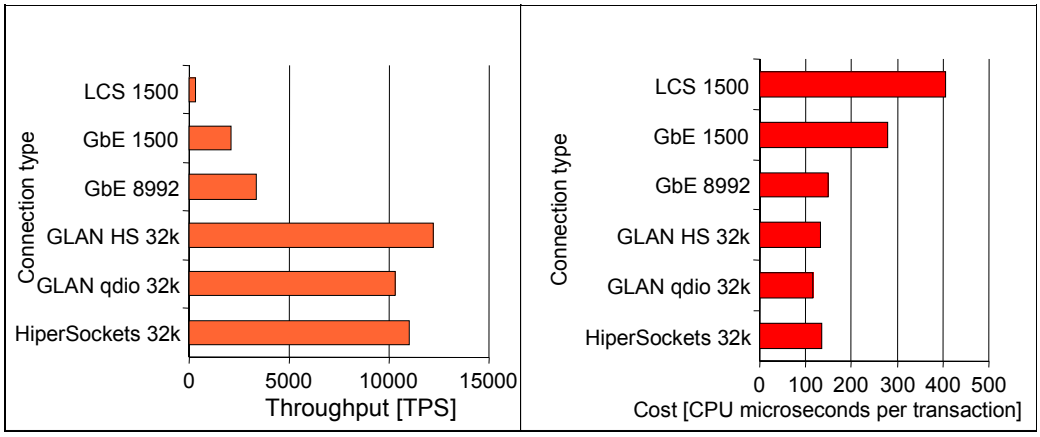

Figure 58 Relative performance of z/VM networking options

For those environments that employ a router to connect the Linux guests to the external network, there are a number of choices. These include using the z/VM TCP/IP stack as a router, using a Linux guest as a router, or eliminating the need for an internal router by using the z/VM Virtual Switch. When using the Virtual Switch, the router function is performed by CP. This means that the CPU time that would have been consumed by a router virtual machine is almost eliminated. This can result in a significant reduction in total system CPU time. The February 2004 version of the *z/VM Performance Report* has documented reductions in CPU ranging from 19% to 33% when a z/VM TCP/IP router was replaced with Virtual Switch (`http://www.vm.ibm.com/perf/docs/zvmperf.html`). Decreases ranging from 46% to 70% were observed when a Linux router was replaced with Virtual Switch. Figure 59 on page 63 shows that the Virtual Switch provides the highest bandwidth at the lowest CPU cost.

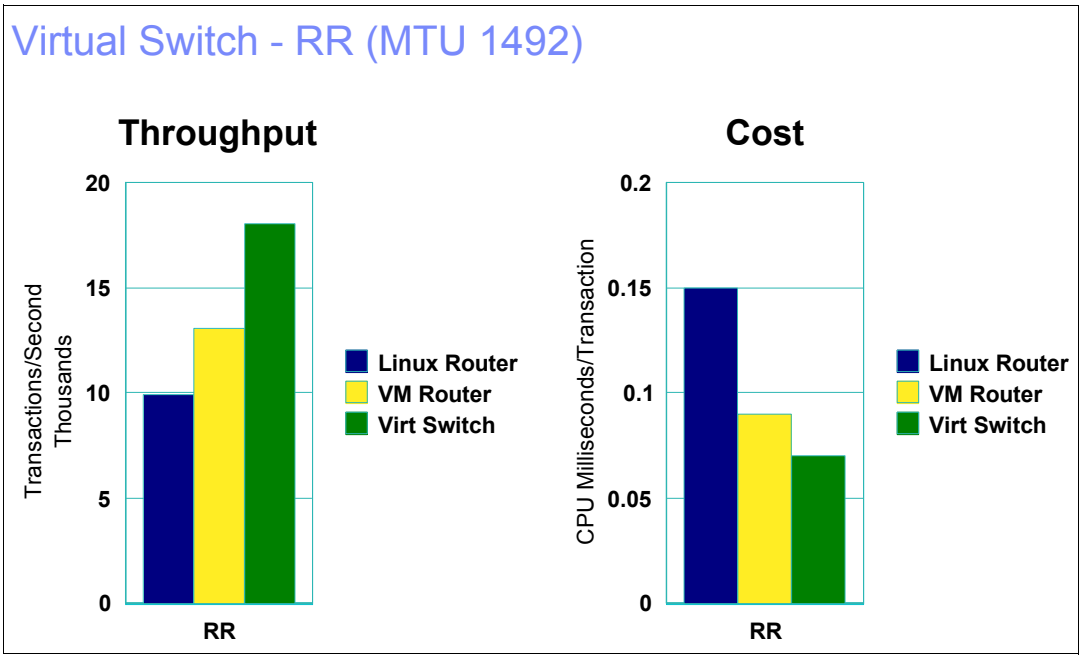

Figure 59 Virtual Switch relative performance

Summary

Networking options for Linux on zSeries have matured significantly since Linux first became available on the mainframe in December 1999. From a hardware perspective over that period, we started with OSA-2 cards, many running at just 10 Mbps. Since then, we have seen the introduction of OSA-Express Gigabit Ethernet, 1000BASE-T, and HiperSockets.

We have seen perhaps even greater advances in networking technology in the virtualized world of z/VM. Originally, apart from a direct connection to a physical interface, the only methods of connecting Linux guests to each other or to an external network were through IUCV or virtual channel to channel adapters. These point-to-point technologies were cumbersome, prone to error, and alien concepts to Linux administrators used to an Ethernet world.

With the introduction of z/VM Guest LAN technology, and in particular the latest Layer 2 Virtual Switch solution, z/VM now has a network virtualization capability that is unparalleled in the industry.

We recommend that you use OSA-Express Gigabit Ethernet or 1000BASE-T for your machine's physical network interfaces. For internal communications, it really depends on what systems are being connected together. For LPAR-to-LPAR communications, we recommend HiperSockets. When multiple Linux guests within a z/VM environment need to communicate with each other, a HiperSockets Guest LAN is one of the best choices. When multiple Linux guests need to communicate within a z/VM LPAR and also with the external network, we recommend that you use a Virtual Switch running in Layer 2 Switching mode.

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

Note that some of the documents referenced here may be available in softcopy only.

- *OSA-2 Implementation Guide (Update)*, SG24-4770
- *OSA-Express Implementation Guide*, SG24-5948
- *FICON CTC Implementation*, REDP-0158
- *zSeries HiperSockets*, SG24-6816
- *Linux on IBM @server zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP-3596
- *Linux on IBM @server zSeries and S/390: VSWITCH and VLAN Features of z/VM 4.4*, REDP-3719
- *Linux on IBM @server zSeries and S/390: Large Scale Linux Deployment*, SG24-6824

Other publications

These publications are also relevant as further information sources:

- *z/VM CP Planning and Administration, Version 5 Release 1.0*, SC24-6083
- *z/VM CP Commands and Utilities Reference, Version 5 Release 1.0*, SC24-6081-00
- *z/VM Connectivity, Version 5 Release 1.0*, SC24-6080-00
- *z/VM TCP/IP Planning and Customization*, SC24-6125
- *OS/390 V2R7.0 OSA/SF User's Guide*, SC28-1855-06
- *VM/ESA OSA/SF User's Guide*, SC28-1992-03
- *Linux for zSeries and S/390, Device Drivers and Installation Commands, October 7, 2004*, LNUX-1313-04

Web sites

These Web sites and URLs are also relevant as further information sources:

- Linux for zSeries device driver documentation

  http://www10.software.ibm.com/developerworks/opensource/linux390/docu/lx24jun03dd04.pdf
- Washington Systems Center Flash *TCP/IP Stack limitation on OSA-Express*

  http://www.ibm.com/support/techdocs/atsmastr.nsf/PubAllNum/Flash10144
- Washington Systems Center Flash *OSA-Express MCL Enhancements - October 2003*

  http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/FLASH10250
- *z/VM Performance Report*, February 23, 2004

  http://www.vm.ibm.com/perf/docs/zvmperf.html
- Linux on zSeries "June 2003 stream" documentation

  http://www.ibm.com/developerworks/opensource/linux390/june2003_documentation.shtml

About the author

Simon Williams is a Consulting IT Specialist and IBM Certified Professional. He is the zSeries Systems Architect for Asia Pacific. He provides technical consulting for z/OS, z/VM, and Linux for zSeries. Before joining IBM, he was a Senior MVS™ Systems Programmer and has been working with mainframe systems since 1988. Simon is an author of *Linux on IBM @server zSeries and S/390: Building SuSE SLES8 systems under z/VM*, REDP-3687, *Building Linux Systems Under IBM VM*, REDP-0120, *Linux on IBM @server zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP-3596. He is the co-author of *Linux on IBM @server zSeries and S/390: ISP/ASP Solutions*, SG24-6299, *Lotus Domino for S/390 Release 5: Installation, Customization and Administration*, SG24-2083, and *Linux on zSeries: Fibre Channel Protocol Implementation Guide*, SG24-6344.

Acknowledgments

The author would like to thank the following people for their invaluable contribution to this Redpaper:

Greg Geiselhart, Mike MacIsaac, Rob van der Heij, Susan Farell, Romney White, Angelo Macchiano, Les Geer, Vic Cross, and Utz Bacher

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

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 email 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

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

ESCON®	IBM®	S/390®
@server®	Multiprise®	z/OS®
FICON®	MVS™	z/VM®
HiperSockets™	Redbooks (logo) ™	zSeries®
ibm.com®	Redbooks™	

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.