Redbooks Paper

Simon Williams

Linux on IBM eserver zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN

Preface

This Redpaper provides information to help readers plan for and exploit Internet Protocol (IP) broadcast support that was made available to z/VM Guest LAN environments with the introduction of the z/VM 4.3 Operating System.

Using IP broadcast support, Linux guests can for the first time use DHCP to lease an IP address dynamically from a DHCP server in a z/VM Guest LAN environment. This frees the administrator from the previous method of having to hardcode an IP address for every Linux guest in the system. This new feature enables easier deployment and administration of large-scale Linux environments.

Objectives

The objectives of this paper are to:

- Review the z/VM Guest LAN environment
- Explain IP broadcast
- Introduce the Dynamic Host Configuration Protocol (DHCP)
- Explain how DHCP works in a z/VM Guest LAN
- Describe how to implement DHCP in a z/VM Guest LAN environment

z/VM Guest LAN

Attention: While broadcast support for z/VM Guest LANs was announced with the base z/VM 4.3 operating system, the user must apply the PTF for APAR VM63172. This APAR resolves several issues which have been found to inhibit the use of DHCP by Linux-based applications running over the z/VM Guest LAN (in simulated QDIO mode).

Introduction

Prior to z/VM 4.2, virtual connectivity options for connecting one or more virtual machines (*VM guests*) was 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; see Figure 1 for a diagram. While 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 two guests to communicate with each other, you must define CTCA device pairs in each guest and couple those devices between the guest machines. Another requirement when using point-to-point connections is the definition of static routing statements in each guest that needs to communicate with any other guest in the system. Finally, another limitation is that 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.

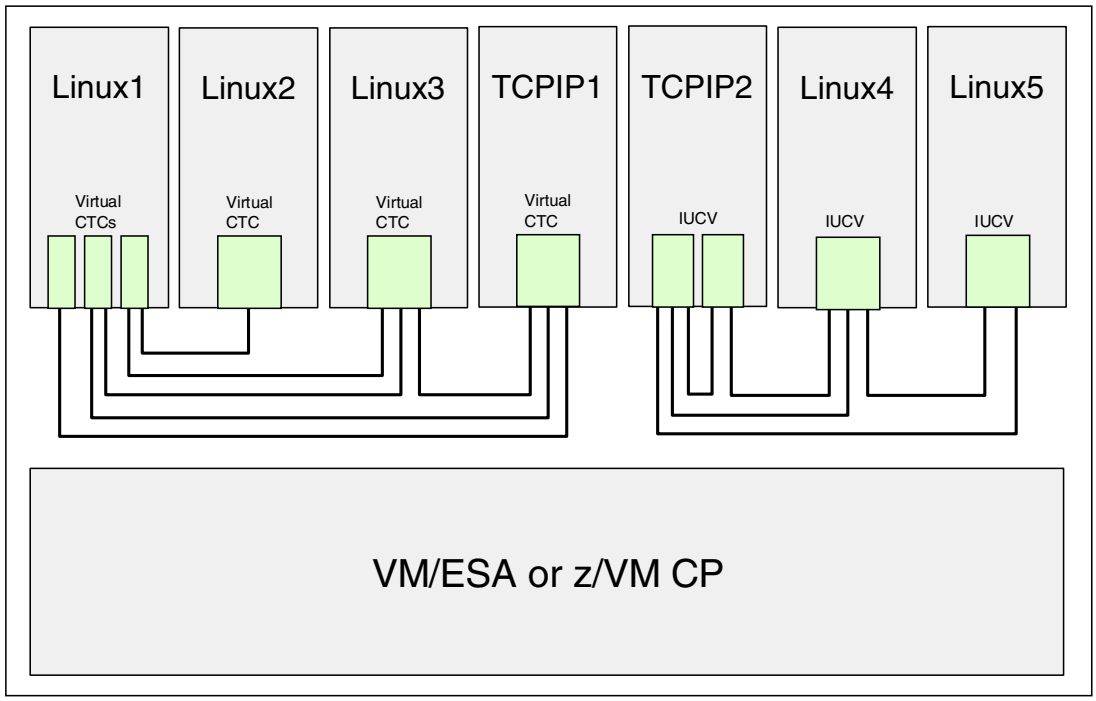

Figure 1 VM point-to-point connections

From z/VM 4.2 onwards, the z/VM Control Program (CP) has been enhanced to provide a feature known as *Guest LAN*. This feature allows you to create multiple virtual LAN segments within a z/VM environment; see Figure 2 on page 3 for an example.

Note: While 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.

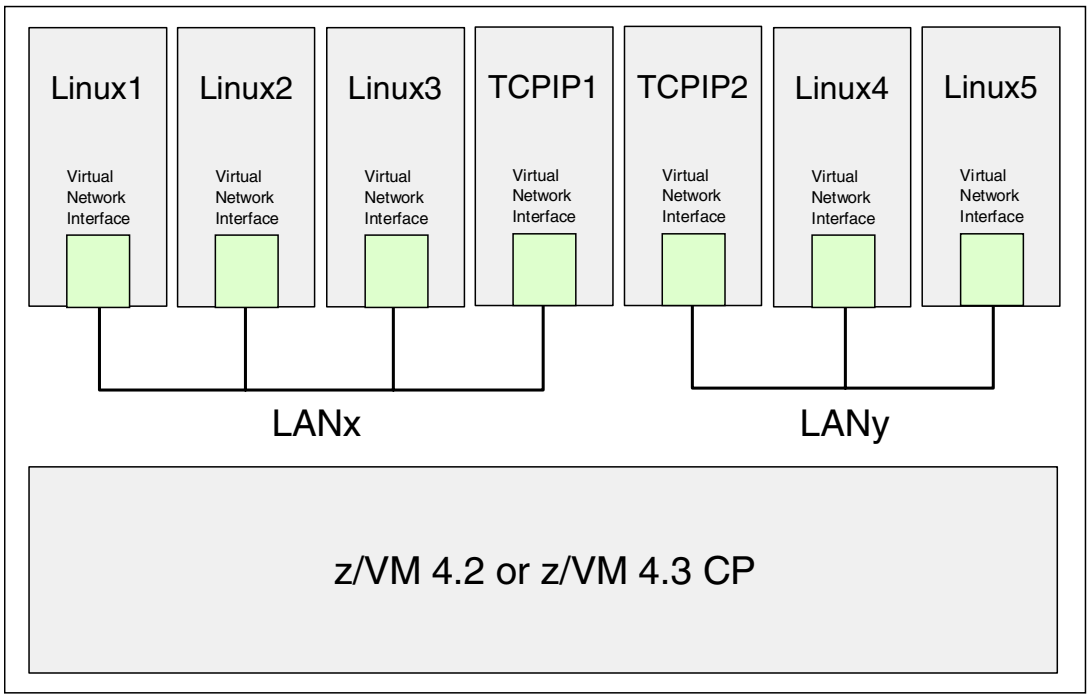

Figure 2 z/VM Guest LAN

There is no architectural limit on the number of Guest LAN segments that you can create. However, you will be limited by the amount of machine resources you have available.

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 then connect this NIC to the LAN and communicate with other guests using standard TCP/IP protocols.

Note: A z/VM Guest LAN is solely contained within a single z/VM operating system environment and thus guests cannot communicate (using their Guest LAN interface) with systems outside of the Guest LAN. Of course, if a Linux guest also has a virtual CTC interface, an IUCV connection, a "real" HiperSocket link, or is the owner of a hardware interface (such as an OSA device), then they can communicate with other systems, providing the necessary addressing and routing is correct.

As z/VM Guest LAN is a virtualization technology, it is not limited to the use of zSeries hardware (that is, the z800 or z900 processors). 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 4.2, the virtual NIC simulated a HiperSockets device, as introduced by the zSeries z900 GA-2 mainframe in late 2001. Being a simulated HiperSockets device meant that the NIC did not have IP broadcast capability.

As of z/VM 4.3 (on which this Redpaper is based), the Guest LAN can be defined to use either simulated HiperSockets devices or simulated QDIO devices. The QDIO device that is

simulated is the OSA Express Ethernet adapter, and as such *QDIO Guest LANs* have IP broadcast support as they are a simulated Ethernet LAN. 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 eth*x* and HiperSockets Guest LANs use a device identifier of hsix.

Guest LAN configuration

While the creation of a Guest LAN is covered in other publications, for ease of reference we review here the steps required to create a Guest LAN and how to connect a Linux guest to that LAN. The steps are:

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. Once the Linux guest has been booted, configure the appropriate device drivers in that guest to connect to the Guest LAN.

Tip: While 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 4 Release 3.0 CP Planning and Administration,* SC24-6043, and *z/VM Version 4 Release 3.0 CP Command and Utility Reference,* SC24-6008.

Create a z/VM Guest LAN

z/VM Guest LANs can be created in one of two ways: either via 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, then 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. For testing purposes, the **`DEFINE LAN`** command is perfectly valid, as it provides the flexibility to dynamically create Guest LANs as required.

Permanent Guest LANs

In order to define a permanent 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 (in this Redpaper, we used *z/VM Version 4 Release 3.0 CP Planning and Administration,* SC24-6043*)*.

The syntax of the **DEFINE LAN** statement is as follows:

DEFINE LAN *lanname* [*operands*]

where:

lanname is a 1- to 8-character alphanumeric name of the z/VM Guest LAN

operands define the characteristics of the z/VM Guest LAN

Operands accepted by the **DEFINE LAN** statement are summarized in Table 1.

Table 1 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.
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[a]. Any other value limits the number of simultaneous connections to a decimal value in the range of 1 to 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 may connect to the LAN. When neither UNRESTricted or RESTricted are specified, then UNRESTricted is the default value[b].
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.

Operand	Description
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, then the default is to GRANT the LAN owner.

a. While an “infinite” number of connections is somewhat optimistic, the actual number will be determined by the physical resources you have defined to your z/VM system. What it does mean, however, is that for practical purposes, you should not run out of connections.
b. We use unrestricted LANs in all of our examples.

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.

Tip: While LIMIT and ACCOUNTING are both valid parameters of the **VMLAN** statement, they cannot coexist within a single **VMLAN** statement. If you want to have definitions for both LIMIT and ACCOUNTING, then create two **VMLAN** statements in the SYSTEM CONFIG file.

Operands accepted by the **VMLAN** command are summarized in Table 2 on page 7.

Table 2 Operands of the VMLAN statement

LIMIT parameter operands	**Description**
PERSistent INFinite | *maxcount*	INFinite means that there will be an infinite number of PERSISTENT[a] 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[b] 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
ACCOUNTing parameter operands	**Description**
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.

a. A Persistent LAN is created when the **DEFINE LAN** SYSTEM CONFIG file statement is used to create a LAN during system initialization, or when a Class B user defines a SYSTEM owned LAN. The term “Persistent” means that a LAN will survive when its owner performs a **LOGOFF** from the system. A Persistent LAN can only be removed by an explicit **DETACH LAN** command. Persistent LANs will not persist across IPLs of the z/VM system unless they are defined in the SYSTEM CONFIG file.

b. A Transient LAN is created when the **DEFINE LAN** command is used to define a general user LAN. A Transient LAN will exist as long as the owning VM user is logged on to the system. Once the owner logs off the VM system, then the transient LAN will be removed by the CP (provided no simulated NICs are coupled to the LAN).

Non-permanent Guest LANs

If we want to define a z/VM Guest LAN that will be available only for the “life of an IPL” (that is, it will be valid only while the system is up and will not be active after the system is next IPLed), then we can use the CP **DEFINE LAN** command. We recommend using this command (as opposed to placing a permanent entry in the SYSTEM CONFIG file) if you require the Guest LAN only for testing purposes.

The syntax of the **DEFINE LAN** command is as follows:

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

where

lanname is a 1- to 8-character alphanumeric name of the z/VM Guest LAN

operands define the characteristics of the z/VM Guest LAN

Operands accepted by the **DEFINE LAN** command are summarized in Table 3 on page 8.

Table 3 Operands of the CP DEFINE LAN command

Operand	Description
OWNERid *ownerid*	Establishes the owner of the LAN. When OWNERid * is specified, the owner is the invoker. This is the default.
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.
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 to 1024 (inclusive).
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 may connect to the LAN. When neither UNRESTricted or RESTricted are specified, then 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.

We can use the **SET LAN** CP command to dynamically adjust the attributes of a Guest LAN. We use the **QUERY LAN** CP command to display the LANs defined to the system.

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-type devices. The NIC can be defined permanently via a User Directory statement or temporarily (for the life of the Guest's session) via 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 **SPECIAL** statement in the z/VM User Directory. The **SPECIAL** statement defines virtual devices that are fully simulated by CP and not connected to real devices at definition time.

The syntax of the SPECIAL statement for Network Interface Cards is as follows:

```
SPEcial vdev [ operands ]
```

where

vdev specifies the base virtual device address for the adapter

operands define the characteristics of the virtual NIC

Operands accepted by the **SPECIAL** command are listed in Table 4.

Table 4 Operands for the SPECIAL 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 Guest LAN of the appropriate type is found when the guest logs on to the system, then the NIC will be automatically coupled to the LAN specified in the *ownerid lanname* parameter (see below).
devs	The number (decimal) of virtual I/O devices to be created for a simulated NIC. If *devs* is omitted, then the default number of devices is 3.
ownerid | * *lanname*	Identifies a Guest LAN segment for an immediate connection to the NIC. If *ownerid* and *lanname* are omitted, then 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 by using an asterisk (*) to represent the user ID of the current virtual machine.

An example CP User Directory entry for a Linux guest that connects to a QDIO Guest LAN is shown in Example 1.

Example 1 User directory entry for a Linux guest - connecting to a QDIO Guest LAN

```
USER LNX23 LNX23 128M 1G G
   INCLUDE IBMDFLT
   IPL 190 PARM AUTOCR
   MACHINE XA
   CONSOLE 0009 3215
   SPECIAL 0700 QDIO 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
```

NIC definition using CP commands

To create a virtual Network Interface Card that will 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

Operands accepted by the **DEFINE NIC** command are listed in Table 5.

Table 5 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 (model 1732-05). 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 (model 1731-01). 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.

Important: We have added the syntax for HiperSockets devices simply for completeness; for broadcast support, keep in mind that we have to use a Guest LAN type of QDIO.

Connect the virtual NIC to the Guest LAN

Now that we have defined the virtual NIC, just like in a real network we now want to connect that device to the LAN. If we had used the **SPECIAL** User Directory statement to define our

NIC, the guest machine would automatically connect to the LAN whenever it logged on to system. 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 as follows:

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

where

vdev specifies the base virtual device address for the adapter

operands defines where to connect the NIC device

Operands accepted by the **COUPLE** command for the purpose of connecting a virtual NIC to a Guest LAN are listed in Table 6.

Table 6 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 **SPECIAL** User Directory statement, consider adding these two commands into your guest’s PROFILE EXEC file so they are automatically executed whenever the guest logs on.

Example of building a z/VM Guest LAN

In Example 2, we demonstrate how to build a z/VM Guest LAN.

Example 2 Steps to build a z/VM Guest LAN

```
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
```

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 Example 3.

Example 3 CP QUERY LAN command

```
QUERY LAN TSTLAN ACTIVE
LAN SYSTEM TSTLAN      Type: QDIO     Active: 1     MAXCONN: INFINITE
  PERSISTENT  UNRESTRICTED  MFS: 8192    ACCOUNTING: OFF
    Adapter Owner: LNX23    NIC: 0700  Name: UNASSIGNED
Ready; T=0.01/0.01 17:36:49
```

Notice from the example that one Linux guest (LNX23) is connected to the Guest LAN.

In order to display information about the virtual NIC that we have defined, we can use the **QUERY NIC** CP command as shown in Example 4. 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 in Example 4, there is no IP addressing information and the number of bytes transmitted and received are both zero. Also, the portname value is set to UNASSIGNED. This tells us that the Linux guest has not started using this device for TCP/IP communications.

Example 4 CP QUERY NIC commands

```
QUERY NIC
Adapter 0700  Type: QDIO      Name: UNASSIGNED  Devices: 3
  Port 0 MAC: 00-04-AC-00-00-01  LAN: SYSTEM TSTLAN      MFS: 8192
Ready; T=0.01/0.01 17:51:51

QUERY NIC DETAILS
Adapter 0700  Type: QDIO      Name: UNASSIGNED  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
  Unassigned Devices:
    Device: 0700  Unit: 000   Role: Unassigned
    Device: 0701  Unit: 001   Role: Unassigned
    Device: 0702  Unit: 002   Role: Unassigned
Ready; T=0.01/0.01 17:51:54
```

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 Example 5.

Example 5 CP QUERY VIRTUAL OSA command

```
QUERY VIRTUAL OSA
OSA  0700 ON NIC  0700  UNIT 000 SUBCHANNEL = 000E
     0700 QDIO-ELIGIBLE
OSA  0701 ON NIC  0700  UNIT 001 SUBCHANNEL = 000F
     0701 QDIO-ELIGIBLE
OSA  0702 ON NIC  0700  UNIT 002 SUBCHANNEL = 0010
     0702 QDIO-ELIGIBLE
Ready; T=0.01/0.01 17:57:19
```

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 shown only for completeness, and you should not follow these steps unless you no longer wish 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. The command is illustrated in Example 6.

Example 6 UNCOUPLE command

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

Remove the virtual NIC from the guest machine

The CP **DETACH NIC** command is used to remove a virtual NIC from a guest machine. 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. Example 7 illustrates the **DETACH NIC** command.

Example 7 DETACH NIC command

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

Remove the Guest LAN

The CP **DETACH LAN** command is used to remove a Guest LAN from the system. 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. Example 8 illustrates the **DETACH LAN** command.

Example 8 DETACH LAN command

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

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 relevant device driver for the virtual Network Interface Card. This driver is called the QETH device driver.

Restriction: The QETH device driver supports OSA Express ethernet cards and the z/VM Guest LAN's simulated QDIO NIC. The level of QETH device driver that we need for DHCP in a Guest LAN was built at revision number 1.260.2.16. This code is available for download from:

http://www10.software.ibm.com/developerworks/opensource/linux390/special_oco_su_2.4.shtml

The QETH device driver we require was built at the 2.4.19 level of the Linux kernel. Our test environment was running the standard SuSE Linux Enterprise Server (SLES) 7 distribution (31-bit) which uses the 2.4.7 Linux kernel. Therefore, we had to upgrade our kernel. For a description of how to upgrade from the 2.4.7 kernel to the 2.4.19 kernel, see Appendix , "Upgrading your Linux kernel" on page 45.

In theory, any Linux distribution available for System/390 or zSeries that can run a 2.4.19 kernel should be able to load the appropriate QETH driver and thus exploit broadcast support in a z/VM Guest LAN. However, due to limited resources, our testing only covered the SuSE distribution.

In the examples that follow we assume you are running a 2.4.19 kernel and have the compatible level of the QETH device driver. The compatible level of this driver is available for download at:

http://www10.software.ibm.com/developerworks/opensource/linux390/special_oco_su_2.4.shtml

The downloaded driver is an Object Code Only (OCO) module, which means there is no requirement to compile this driver. For an example of how to copy this driver into your Linux environment, see "Copy OCO modules" on page 49.

To determine if you have the correct level of the QETH device drive, check the revision number. If your revision numbers are lower than those in Example 9, your QETH device driver will not have support for IP broadcast.

Example 9 Checking the driver revision level

```
lnx20:~ # strings /lib/modules/2.4.19/kernel/net/qeth.o | grep Revis

QDIO OSA-Express driver ($Revision: 1.260.2.16 $)
qeth S/390 OSA-Express driver ($Revision: 1.260.2.16 $/$Revision: 1.86.2.2 $/$Revision:
1.31.4.1 $:IPv6:VLAN)
```

Before we load the QETH driver, we must first pass configuration information about the virtual NIC to the Linux kernel's Channel Device Layer.

Channel Device Layer

The Channel Device Layer provides a common interface to zSeries and System/390 channel attached devices. This interface can be used to configure (among other things) the simulated QDIO Network Interface Card that we have defined and connected to the z/VM Guest LAN.

Tip: For a detailed review of the Channel Device Layer and the syntax of the QETH device driver configuration, refer to the appropriate level of the device drivers manual. For this Redpaper, we used *Linux for zSeries and S/390 Device Drivers and Installation Commands, July 31, 2002, Linux Kernel 2.4* LNUX-1303-01.

The document can be downloaded from the following URL:

http://www10.software.ibm.com/developerworks/opensource/linux390/docu/lnuxdd01.pdf

There are three ways to pass configuration settings to the Channel Device Layer.

1. By redirecting the settings to the /proc/chandev file. This file holds the current configuration of the channel device layer.
2. By adding the settings to the /etc/chandev.conf file.
3. By using the 'chandev=' keyword on the Linux Kernel boot command line.

For testing purposes we used the first method; that is, we redirected the configuration settings into the /proc/chandev file. Once we were satisfied that the configuration was correct, we automated the process so that the driver was loaded at Linux boot time by adding our settings to the /etc/chandev.conf file.

The configuration settings we redirected to the /proc/chandev file are shown in Example 10.

Example 10 Redirecting QETH device driver settings to /proc/chandev

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

Table 7 describes each of these settings.

Table 7 QETH device driver parameters

Parameter	Description
noauto	This stops auto-detection of channel devices.
qeth-1,0x0700,0x0701,0x0702,0,0	qeth-1 This is 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, then the next device to be defined would be qeth2.
	0x700 This is the read subchannel address.
	0x701 This is the write subchannel address.
	0x702 This is 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.

Parameter	Description
`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` This identifies the port name as TSTLAN. This is the name of our z/VM Guest LAN.

Tip: Strictly speaking, the portname parameter (in our case, TSTLAN) is not required for z/VM Guest LANs (if the device were a real OSA-Express interface, the portname would be required and *must* match the name defined in the adapter hardware definition).

However, we recommend that you specify the portname whenever you configure a z/VM Guest LAN interface. The name you choose is unimportant, but for consistency we recommend you use the z/VM Guest LAN segment name.

A partial extract from the /proc/chandev, showing only the changes after we redirected our settings for the virtual NIC, is shown in Example 11.

Example 11 Partial contents of /proc/chandev after redirecting the QETH device driver settings

```
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
```

1. We chose to use `noauto`. As we did not specifiy any devices, autodetection has been turned off for all devices.
2. Our three devices (`0x700,0x701,0x702`) have been initialized. They are of channel type `0x10`, meaning 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`.
3. The z/VM Guest LAN that we are connected to has a portname 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 by using the **`lsmod`** command, as shown in Example 12.

Example 12 lsmod command output

```
lnx20:~ # lsmod
Module                  Size  Used by
qeth                  149232   1 (autoclean)
qdio                   40128   1 (autoclean) [qeth]
ipv6                  254016  -1 (autoclean) [qeth]
```

We see the modules are loaded - this is because we used an OSA Express card for connectivity to the Linux guest.

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, because doing so would disrupt connectivity to the devices which rely on those modules.

Instead, once 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
```

An example illustrating the output on the Linux guest's 3270 session when loading the device drivers is shown in Example 13 on page 18.

Example 13 Load the qdio and qeth 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
```

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 automatically load those other modules as well. So in our example, a **modprobe qeth** would load both the qeth and qdio modules.

In Example 14, we double-check that the device drivers were loaded.

Example 14 Output of the lsmod command

```
lnx20:~ # lsmod
Module                  Size  Used by
qeth                  149232   1 (autoclean)
qdio                   40128   1 (autoclean) [qeth]
ipv6                  254016  -1 (autoclean) [qeth]
```

If we run a CP **QUERY NIC DETAILS** command from the Linux guest's 3270 session, we see in Example 15 on page 19 that the NIC has now been linked to the Guest LAN at both a CP and Linux device driver level.

Example 15 QUERY NIC DETAILS example

```
#CP 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
```

1. The Name field is the *portname* 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 /proc/chandev and loading the device drivers, we see that we now have a value of TSTLAN, which matches the name of our Guest LAN.

Note: As a test, we changed the portname defined through /proc/chandev to something completely different. We found that we were still able to communicate to other Linux systems on the Guest LAN.

With the assumption that 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. Output of this command is shown in Example 16.

Example 16 Output of the ifconfig -a 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
```

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. However, since we now have “broadcast support”, we chose to use a DHCP client to lease an address from a DHCP server machine that is running on the Guest LAN. This is covered in “Implementing DHCP in a z/VM Guest LAN” on page 34.

Now that we have tested the new network interface, we can 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 shown in Example 17.

Example 17 Entry in /etc/chandev.conf for the virtual NIC

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

An explanation of the parameters is covered in Table 7 on page 15.

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 can't 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
```

IP addressing

The unit of transfer in an IP network is called an *IP datagram*. It consists of an IP header and data relevant to higher level protocols (for example, TCP or UDP). One of the components of the IP header is the destination machine's IP address. Depending on the method of IP datagram delivery, this address could be one of the following types:

1. Unicast address
2. Broadcast address
3. Multicast address
4. Anycast address

In this Redpaper, we are interested in the *broadcast* address as it enables us to implement DHCP in a z/VM Guest LAN. However, for completeness, we will review the four different address types illustrated in Figure 3 on page 21.

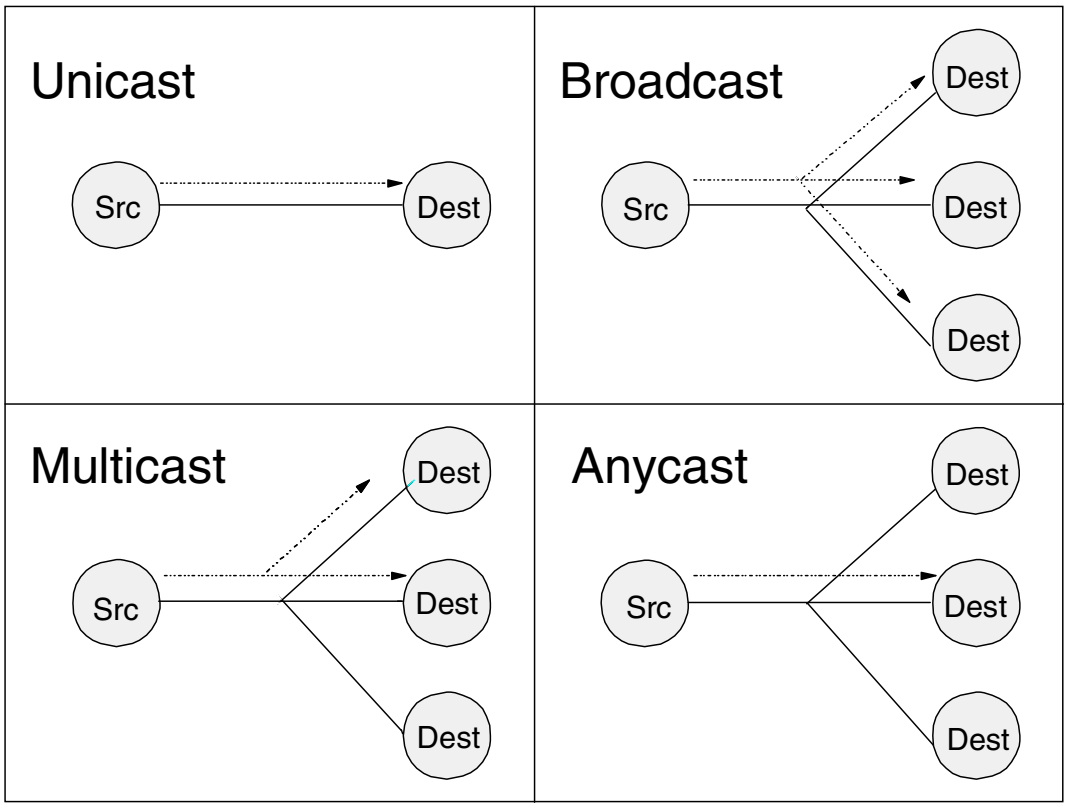

Figure 3 IP packet delivery modes (showing delivery from source to destination)

Unicast

Most destination IP addresses are to a single recipient; this is called a *unicast* address. Unicast connections specify a one-to-one relationship between a single source and a single destination.

Broadcast

A *broadcast* address is a special type of IP address. Using broadcast, a message can be sent from one machine to all other machines on a LAN segment. The sending machine can do this without having to know the IP addresses of all the other machines.

This becomes useful when a machine wishes to send information to all other machines on a LAN (to let all machines know of an update to a critical piece of networking information such as a change of DNS address, for example). Broadcast is also useful when a machine does not know the address of its neighbors, but wishes to request services from a machine on the LAN. A classic example of this is the Dynamic Host Configuration Protocol (DHCP). Using DHCP, a machine will send out a broadcast request asking “somebody” (it doesn’t know whom) to lease it an IP address.

As discussed previously, IP Broadcast support was made available for z/VM Guest LANs with the introduction of z/VM 4.3 and APAR VM63172. There are several different types of broadcast addresses; however, the one used by a DHCP client is called the *Limited broadcast address*. This uses the address 255.255.255.255 (all bits set to 1 in the IP destination address). It refers to all hosts on the local subnet and is recognized by every host. Routers do not forward this packet.

One exception to this rule is called *BOOTP forwarding*. The BOOTP protocol (a forerunner to DHCP) uses the limited broadcast address to allow a diskless workstation to contact a boot server. BOOTP forwarding is a configuration option available on some routers. Without this facility, a separate BOOTP server is required on each subnet.

Multicast

If an IP datagram is broadcast to a subnet, it is received by every host on the subnet. Each host processes the packet to determine if the target protocol is active. If it is not active, the IP datagram is discarded. Multicasting avoids this overhead by selecting destination groups. Each destination group is represented by a special IP address known as a *multicast address*. For each multicast address, a set of one or more hosts are listening for packets addressed to that destination address. This set of hosts is called the *host group*. Packets sent to a multicast address are forwarded only to the members of the corresponding host group. Multicast enables one-to-many connections (see Figure 3 on page 21).

Anycast

Sometimes, the same IP services are provided by different hosts; for example, a user wants to download a file via FTP and the file is available on multiple FTP servers. Hosts that implement the same service provide an anycast address to other hosts that require the service. Connections are made to the first host in the anycast address group to respond. This process is used to guarantee the service is provided by the host with the best connection to the receiver.

An introduction to DHCP

Dynamic Host Configuration Protocol (DHCP) is a TCP/IP protocol which allows clients to obtain IP network configuration information (including an IP address) from a central DHCP server. The DHCP server controls whether the address it provides to a client is allocated permanently or is leased for a period of time. When a client receives a leased address, it must periodically request that the server re-validate the address and renew the lease. The predecessor to DHCP was called the Bootstrap Protocol (BOOTP). BOOTP did not support address leasing.

Note: DHCP specifications are described by RFC 2131 “Dynamic Host Configuration Protocol” and RFC 2132 “DHCP options and BOOTP Vendor Extensions”, which are available from the Internet at:

`http://www.ietf.org/rfc.html`

The following components make up a DHCP environment:

1. DHCP clients

   In our example, these are Linux guest machines that run the DHCP client program.

2. DHCP Server

   This is one or more machines that provide the IP address and configuration information to DHCP clients on the network. In our example, the DHCP server will be a Linux guest that has a static IP address.

3. DHCP Relay Agent

   Relay Agents are used in IP routers to forward information between DHCP clients and servers on different subnets. DHCP Relay Agents eliminate the need for a DHCP server on each subnet. We will not be implementing a DHCP Relay Agent in our examples.

These components are illustrated in Figure 4.

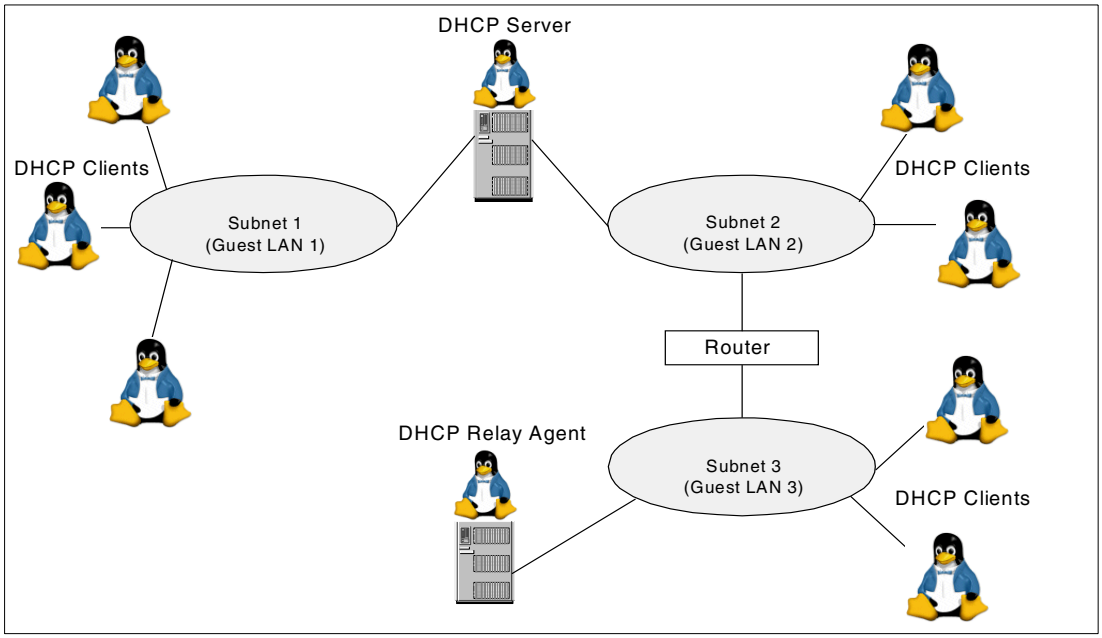

Figure 4 DHCP network components

DHCP defines three IP address allocation policies:

- Dynamic

  A DHCP server assigns a temporary, leased IP address to a DHCP client.

- Static

  A DHCP server assigns a static, predefined address that is reserved for a specific DHCP client.

- Permanent

  A DHCP server assigns a permanent IP address to a DHCP client. No lease renewal is required.

Restriction: In a z/VM Guest LAN environment, the MAC address of the virtual Network Interface Card is assigned randomly (within certain predefined boundaries). This is because the MAC address is not associated with a physical card, but instead with a simulated device. This means that a Linux guest may not get the same MAC address for its virtual NIC every time it logs on to z/VM.

Typically, DHCP servers use the MAC address of a client to determine if that machine will be given a dynamic or static/permanent IP address. As we cannot guarantee that a Linux guest will always have the same MAC address, we need to use another means of identifying client machines. We can do this using a unique client identifier (discussed in “Fixed IP address allocation” on page 42).

Obtaining configuration information

DHCP clients can obtain an IP address and other configuration information through a request dialog with a DHCP server, as illustrated in Figure 5 on page 24.

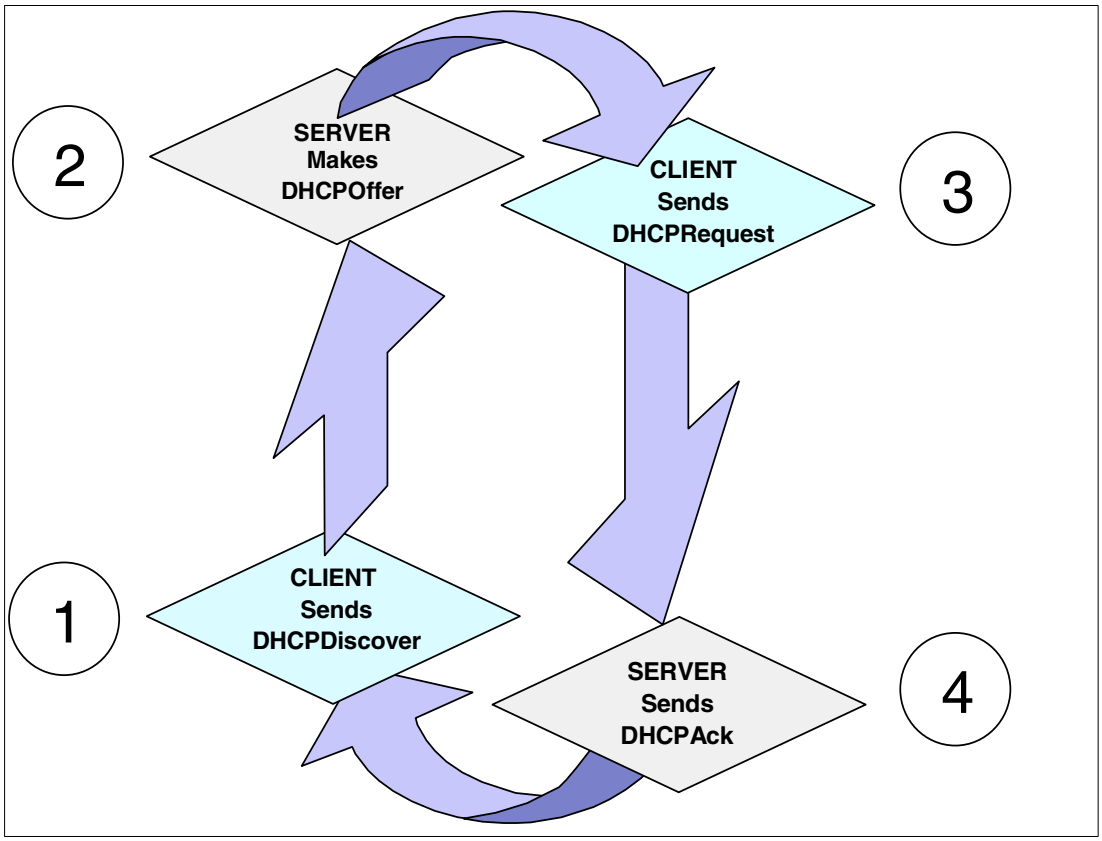

Figure 5 DHCP cycle

We now walk through an example of a DHCP client requesting IP configuration information from a DHCP server.

The DHCP client machine (in our case, a Linux guest) *broadcasts* a DHCPDISCOVER message on its local subnet; see Figure 6 on page 25 for an example. This message is used to request an IP address from any DHCP server that might be listening.

The DHCPDISCOVER message may include other options requested by the client, such as a subnet mask, domain name server, and domain name. The client can suggest specific values for some options. Optionally, if routers on the network are configured to forward DHCP and BOOTP messages, the broadcast message is forwarded to DHCP servers on the attached subnets

Note: Prior to z/VM 4.3 with APAR VM63172, the DHCPDISCOVER message could not have been broadcasted across the Guest LAN. That is why DHCP is only capable of running from that level of z/VM onwards.

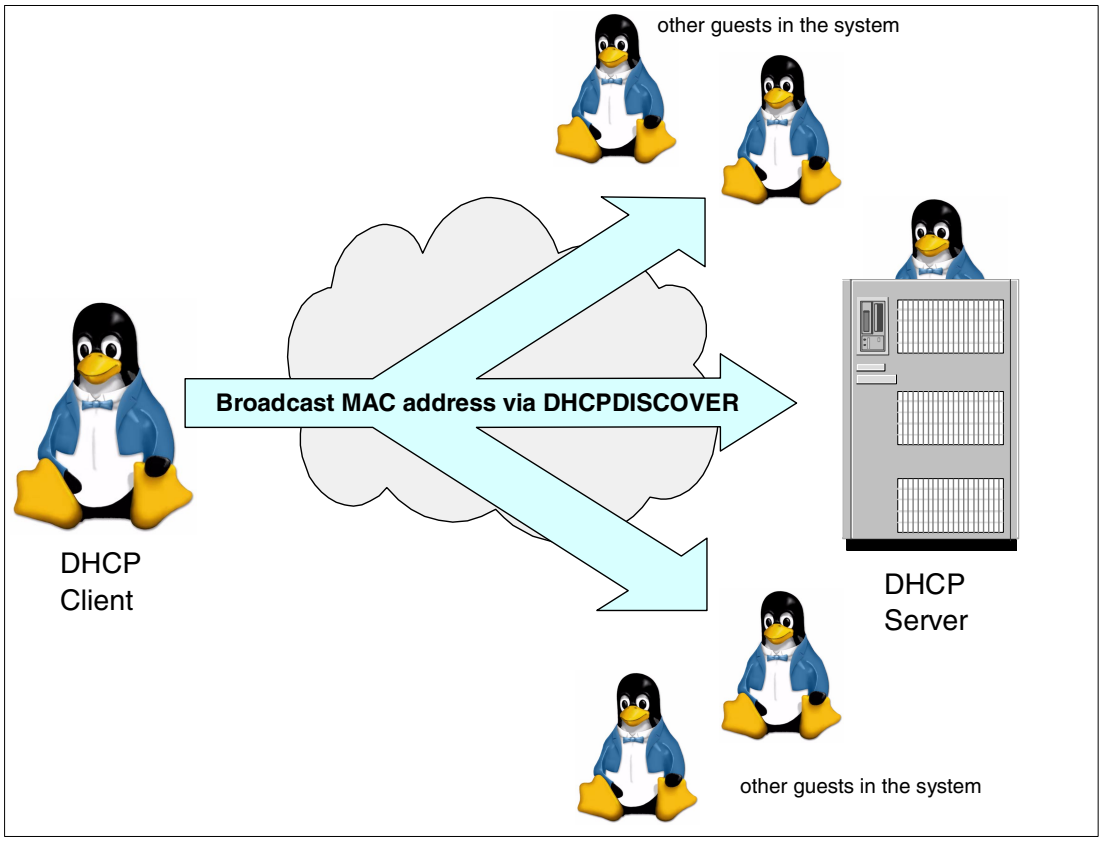

Figure 6 DHCPDISCOVER message broadcast

Each DHCP server (in our example, this is another Linux system on the Guest LAN) will respond with a DHCPOFFER message that includes an available IP address and other configuration options; see Figure 7 on page 26 for an example.

The DHCP server checks its configuration file to see if it should assign a static or dynamic address to this client. In the case of a dynamic address, the server selects an address from the address pool, choosing the least recently used address. An address pool is a range of IP addresses available for leasing to clients.

In the case of a static address, the server uses a client-identifier statement from the DHCP server configuration file to assign options to the client. Upon making the offer, the DHCP server reserves the offered address to prevent the same address being offered to other clients in the event of further DHCPDISCOVER messages being received before the first client has completed its configuration.

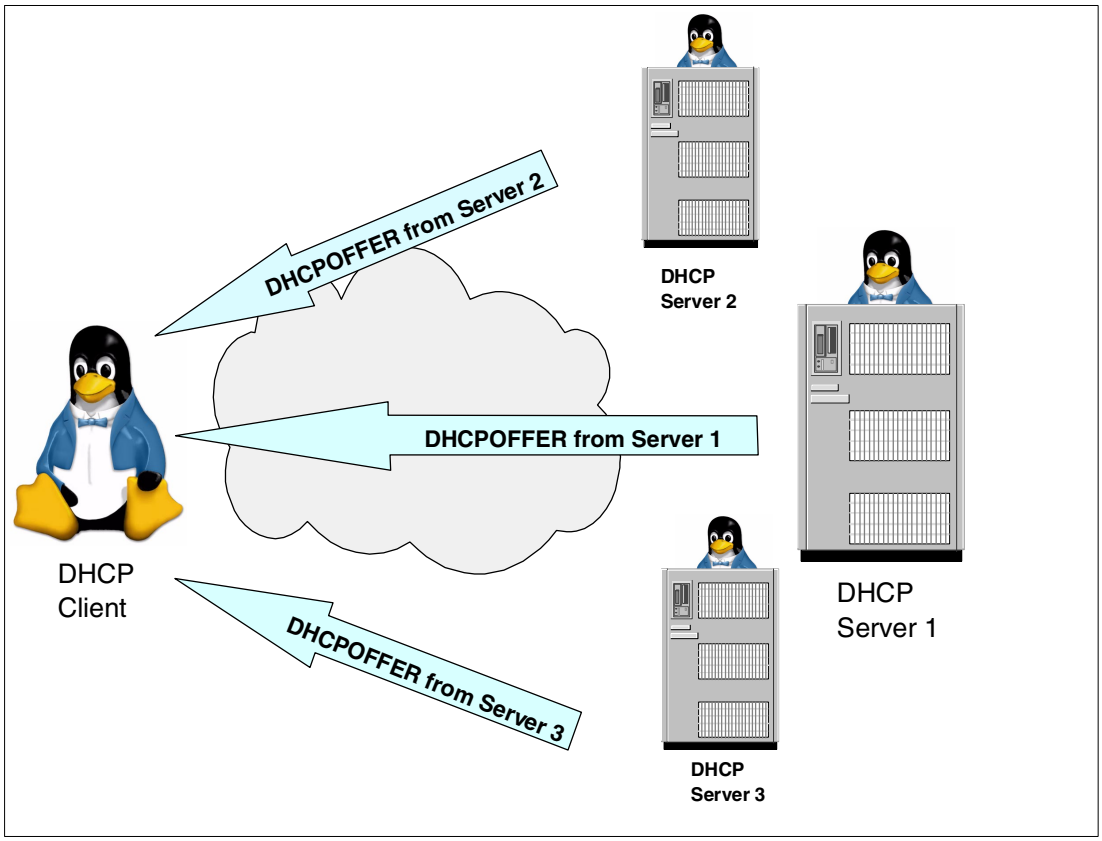

Figure 7 DHCPOFFER message

The DHCP client receives a DHCPOFFER message from one or more servers. The client chooses an address based on the configuration parameters that were offered. The client then broadcasts a DHCPREQUEST message (see Figure 8 on page 27) that includes the server identifier option to indicate which offer it has selected.

If no offers are received, and if the client has knowledge of a previous network address, the client may reuse that address if its lease is still valid. When the lease expires, it must then go through the DHCPDISCOVER process again.

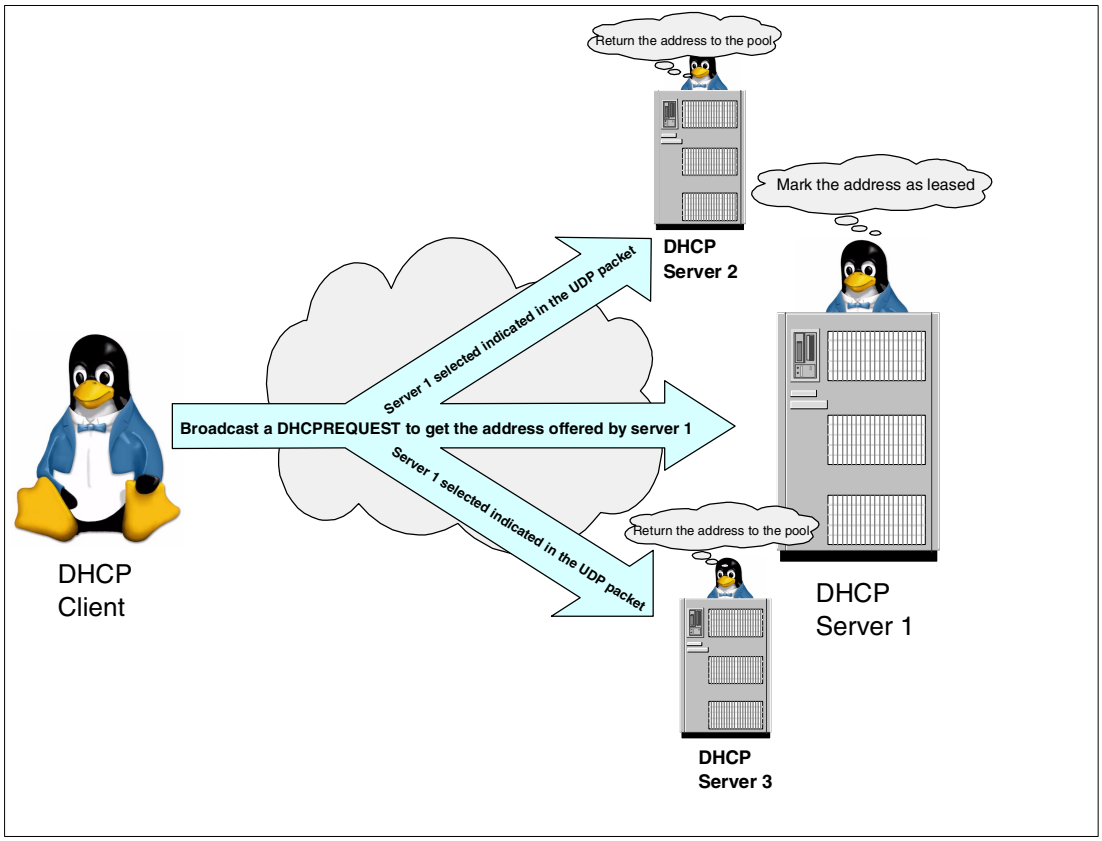

Figure 8 DHCPREQUEST message

If a server receives a DHCPREQUEST message indicating that the client has accepted the server's offer, the server marks the address as leased. If the server receives a DHCPREQUEST message indicating that the client has accepted an offer from a different server, the server returns the address to the available pool. If no message is received within a specified time, the server returns the address to the available pool. The selected DHCP server sends an acknowledgment which contains additional configuration information to the client (DHCPACK message); see Figure 9 on page 28 for an example.

The client determines whether the configuration information is valid. Upon receipt of a DHCPACK message, the DHCP client sends an Address Resolution Protocol (ARP) request to the supplied IP address to see if it is already in use. If it receives a response to the ARP request, the client declines (DHCPDECLINE message) the offer and initiates the process again. Otherwise, the client accepts the configuration information. By accepting a valid lease, the client enters a BOUND state with the DHCP server, and proceeds to use the IP address and options.

For DHCP clients that request options, the DHCP server typically provides configuration information that includes subnet mask, domain name server (DNS), domain name, and gateway information.

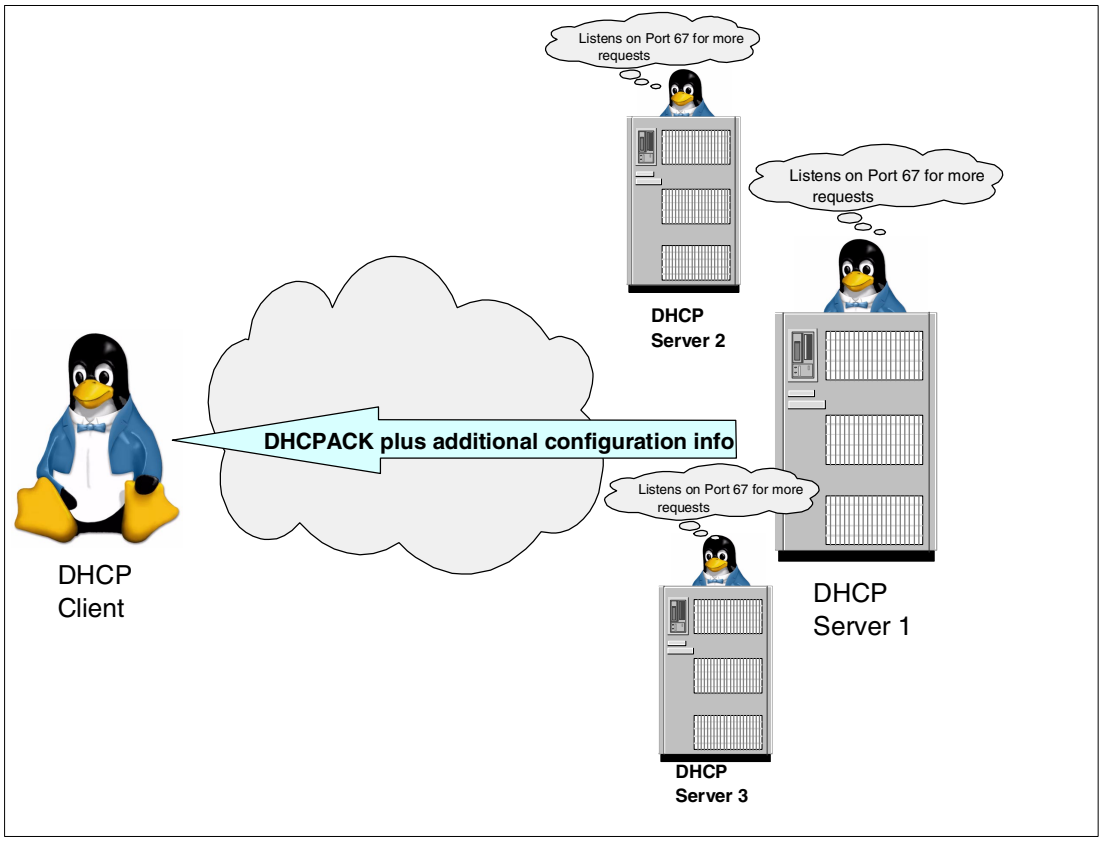

Figure 9 DHCPACK message

Lease renewal

The DHCP client keeps track of how much time is remaining on the lease. At a specified time prior to the expiration of the lease (usually when half of the lease time has passed), the client sends a renewal request containing its current IP address and configuration information to the leasing server. If the server responds with a lease offer, the DHCP client's lease is renewed.

If the DHCP server explicitly refuses the request, the DHCP client may continue to use the IP address until the lease time expires and then initiate the address request process, including broadcasting the address request. If the server is unreachable, the client may continue to use the assigned address until the lease expires.

DHCP relay agent

When a DHCP client and a DHCP server are located in different physical subnets, a DHCP Relay Agent can be used to forward any DHCP request that it receives on its subnet or from other subnets in the direction of the DHCP server. When the DHCP client creates a DHCPDISCOVER packet, a field in the message called RELAY AGENT is set to zeros.

When the Relay Agent intercepts a DHCPDISCOVER message, it examines the RELAY AGENT field. If the RELAY AGENT field is all zeros (that is, the DHCP message comes directly from the DHCP client), the Relay Agent will write its own IP address in this field and then forward the packet to the next hop and increase the hop count.

If the RELAY AGENT field is not zero (that is, the DHCP message comes from another Relay Agent), the Relay Agent just forwards the packet to the next hop and increments the hop count, without modifying the RELAY AGENT field. This process is repeated until the packet reaches the DHCP server.

When the DHCP server sends the DHCPOFFER, it sends the message back to the first Relay Agent and the Relay Agent forwards it to the originating client. Once the client receives an IP address, the communication is direct between the DHCP server and the DHCP client.

DHCP in a z/VM Guest LAN

A QDIO Guest LAN is a virtualized implementation of a standard Ethernet LAN; the difference is that the Guest LAN resides solely within the z/VM system and has no physical components. DHCP behaves the same in a Guest LAN as it does in a “real” LAN.

To get a better understanding of DHCP client/server interaction, we use the network protocol analyzer tool Ethereal to monitor our DHCP server’s eth2 interface, which is connected to the the Guest LAN “TSTLAN”. Ethereal is an Open Source software package available from:

```
http://www.ethereal.com
```

The syntax to start Ethereal in interactive mode is:

```
ethereal -i eth2 -k
```

Our example environment is shown in Figure 10.

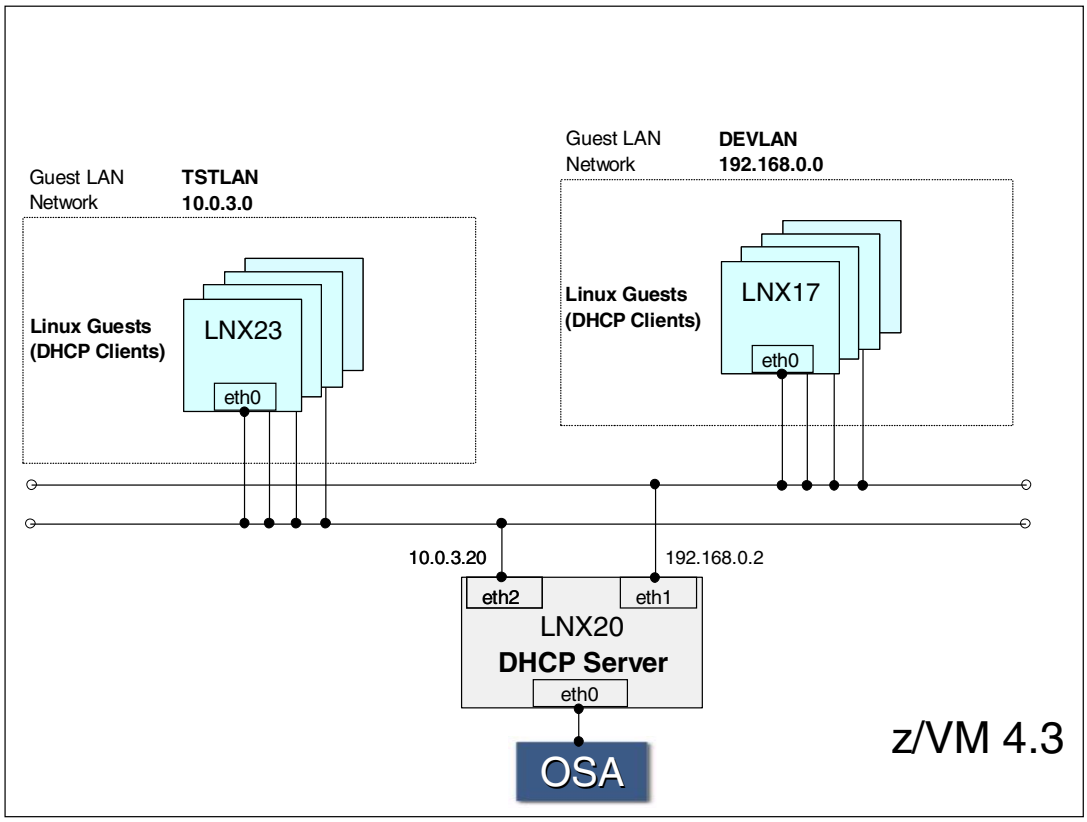

Figure 10 Example network environment

In our example, the Linux guest (LNX23) is booted. Part of the boot process is to configure the network interface. LNX23 has one network interface called eth0. This interface has been set up to request a dynamic IP address using the DHCP client program dhcpcd.

As discussed in “Obtaining configuration information” on page 23, when a DHCP client requests an IP address, it sends out a *broadcast* DHCPDISCOVER message; see Figure 11 for a dump of this message.

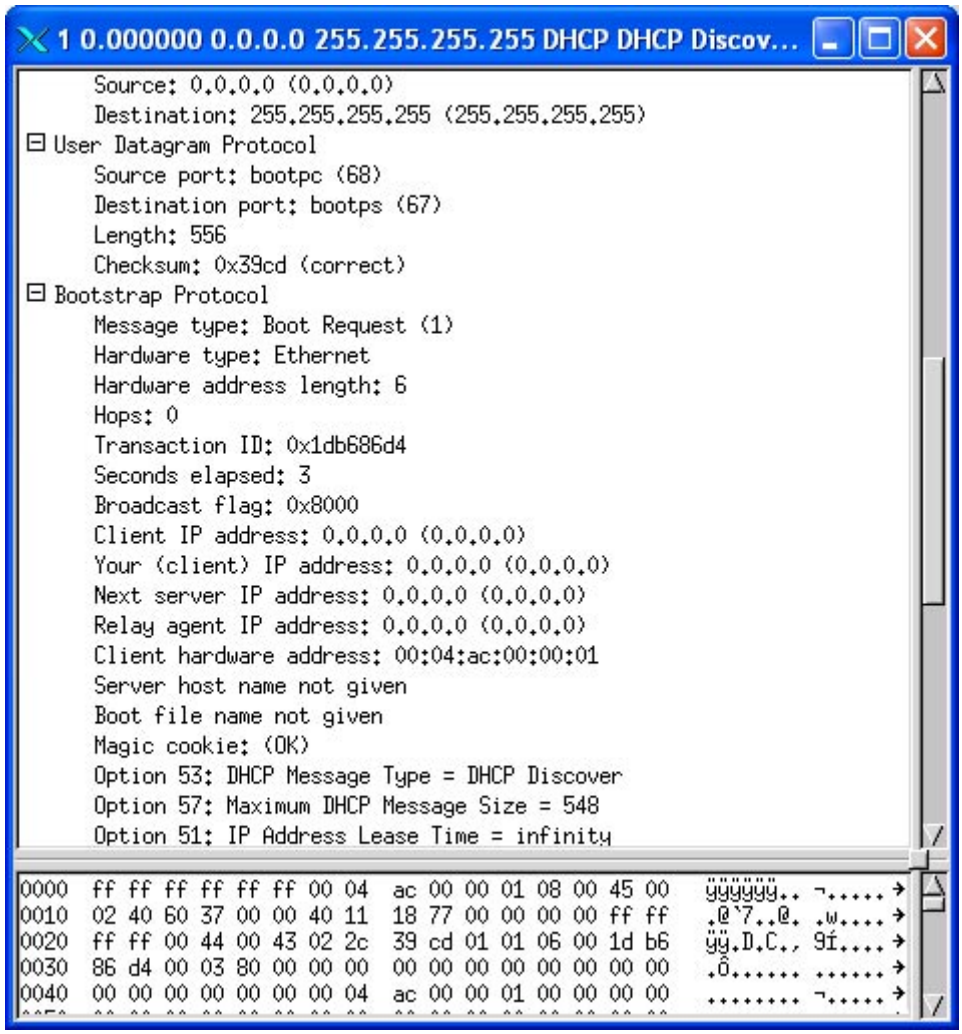

Figure 11 Ethereal output of DHCPDISCOVER message

Note: The DHCP protocol is based on the BOOTP protocol, which is why you see “Bootstrap Protocol” mentioned in the output.

From this output, we can see that the source address is 0.0.0.0, as the requesting system does not yet have an IP address. The destination IP address is 255.255.255.255, which is the limited broadcast address. Because the client has broadcast the DHCPDISCOVER message, all machines in the subnet will receive the message and any DHCP servers that are listening (on port 67) will respond.

When the client sends a request to a DHCP server, it sets a transaction ID (in our example, 0x1db686d4). The server copies that transaction ID from the request into the response that it

sends to the client. When the client reads the response, it checks the transaction ID. If the ID matches its earlier request, the client processes the message. If there is no match, the message is discarded. While theoretically possible, it is highly unlikely that two DHCP clients will simultaneously choose the same transaction ID and thus potentially try to lease the same IP address from the DHCP server.

The *broadcast flag* is set on (0x8000) because our client is running the **dhcpcd** command with the **-B** parameter. With this parameter, the client requests that the response from the DHCP server be sent as a broadcast message.

Once the DHCP server has received the DHCPDISCOVER message from the client, it will respond with a DHCPOFFER. Figure 12 shows the DHCPOFFER output from the Ethereal packet sniffer.

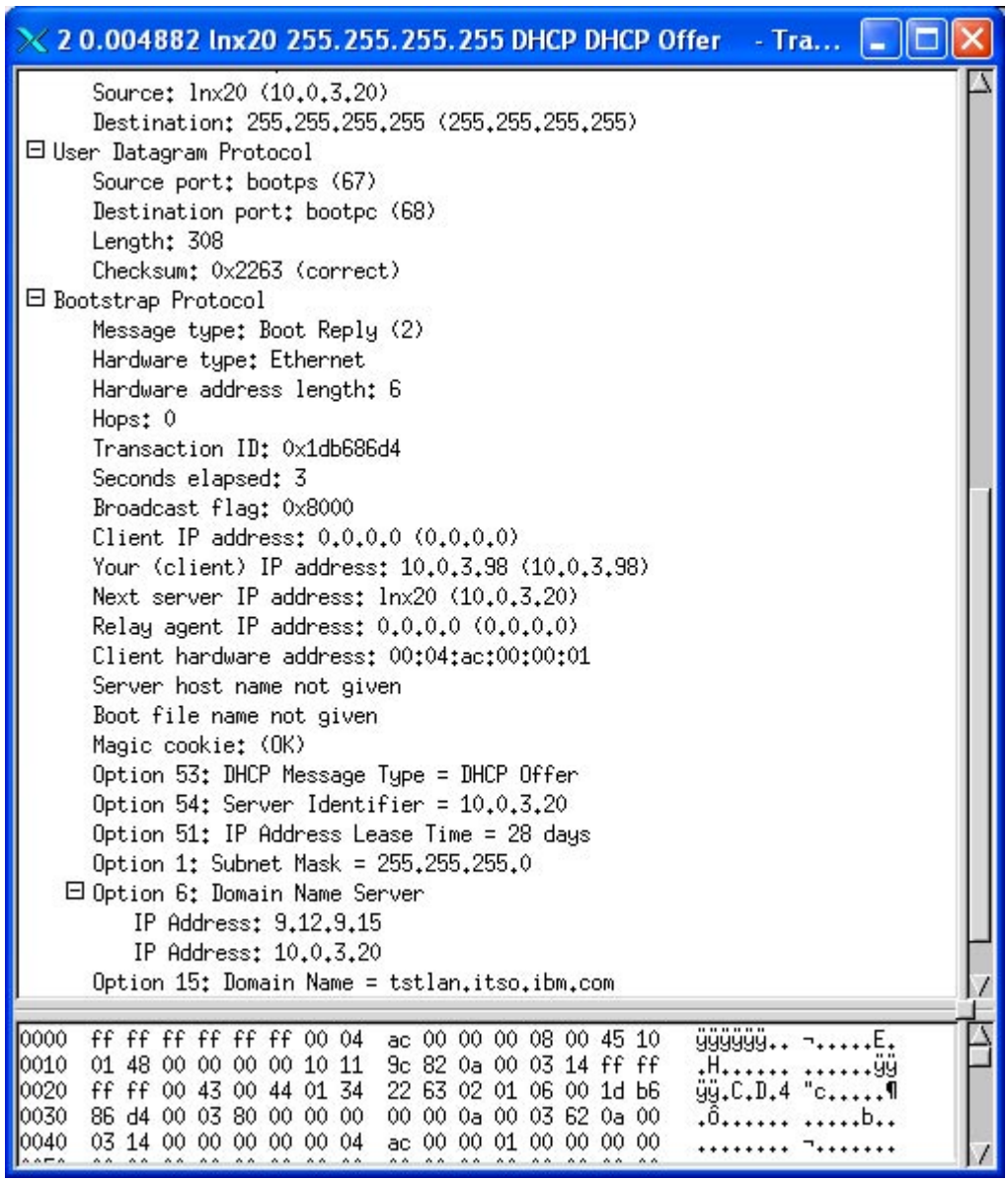

Figure 12 Ethereal output of DHCPOFFER message

From this output, we can see that the source of the DHCPOFFER is 10.0.3.20, which is our DHCP server. The destination of the DHCPOFFER is the limited broadcast address 255.255.255.255. Because the client specified **dhcpcd -B**., the server broadcasts the offer.

Note: If the DHCP client had not requested a broadcast message, the server's offer would have been sent as a unicast message. This does not work in the current Guest LAN implementation. Obviously, there is a minor overhead in sending the DHCPOFFER to every DHCP client listening on the subnet; however, we believe that this is negligible when compared to the value of using DHCP in a Guest LAN.

The client is being offered IP address 10.0.3.98 with a lease time of 28 days. For production Linux systems, you would probably want a lease time set to several weeks or months. The client is also being supplied with a subnet mask, DNS addresses, and domain name.

Once the client has received a DHCPOFFER from a server, it will send a DHCPREQUEST message to that server to ask for the lease. Figure 13 shows the output from Ethereal.

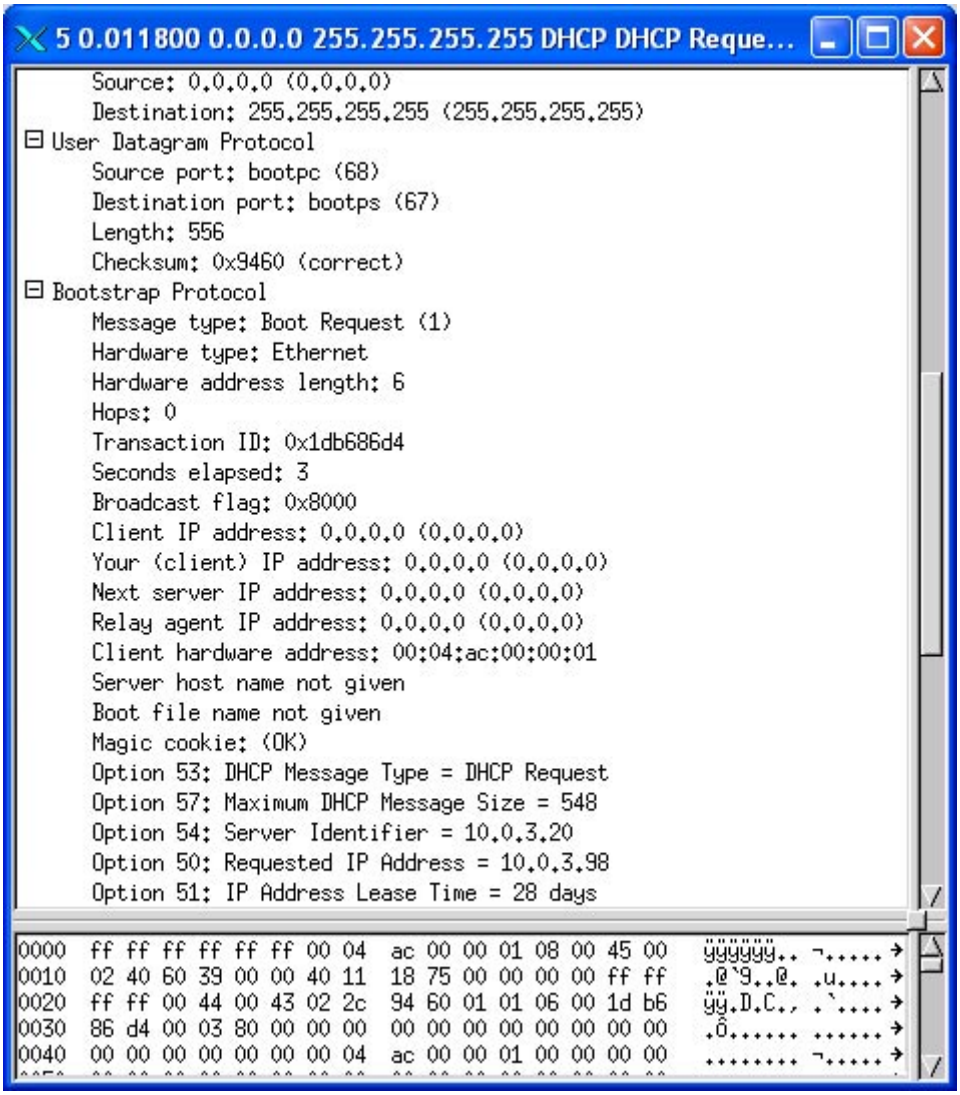

Figure 13 Ethereal output of DHCPREQUEST message

From the output, we see that the client has requested IP address 10.0.3.98 (this being the IP address offered by the server). The DHCPREQUEST is sent as a broadcast message, as there may have been multiple servers that offered IP addresses to the client. Each server will check the DHCPREQUEST message. If the IP address in the message is one that it has offered, the server checks to determine if that address is still available. If so, the server grants the request by sending a DHCPACK message to the client, as seen in Figure 14.

If a DHCP server determines that the IP address in the DHCPREQUEST message is not one which it has offered, it discards the message and makes the address available to offer other requestors.

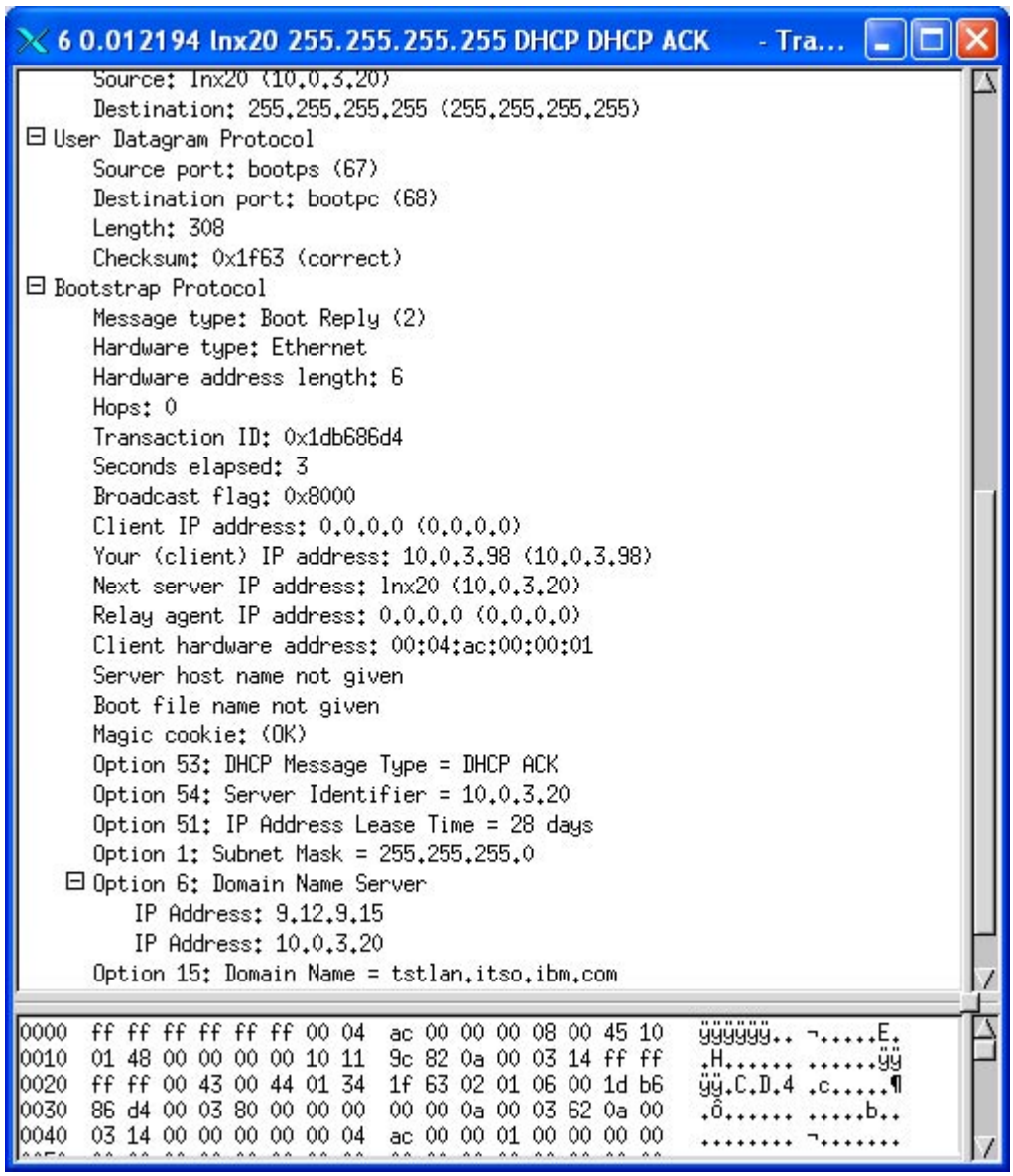

Figure 14 Ethereal output of DHCPACK message

Prior to accepting the configuration details contained in the DHCPACK, the client needs to determine if the IP address being offered is valid. The client does this by sending an Address Resolution Protocol (ARP) request to the IP address to see if it is already in use; another

system may have manually configured an interface to use the IP address without the knowledge of the DHCP server.

If the client receives a response to the ARP request, the client declines (DHCPDECLINE message) the offer and initiates the process again. The safest way to avoid this problem is to ensure that only authorized administrators have access to change networking settings.

Implementing DHCP in a z/VM Guest LAN

Important: Examples in this section use the DHCP client and server packages that are distributed with the SuSE Linux Enterprise Server (SLES) 7. Specifically we used:

Client

```
dhcpcd - DHCP Client Daemon v.1.3.19-pl9
```

available from:

```
http://www.phystech.com/download/
```

Server

```
dhcpd - Internet Software Consortium DHCP Server V3.0rc12
```

available from:

```
http://www.isc.org
```

Now that we have reviewed some of the theory behind DHCP and have seen a “real-world” example of a DHCP client/server conversation, we discuss how to implement DHCP in a z/VM Guest LAN environment.

In this section, we review the following:

- DHCP server configuration
- DHCP client configuration
- Dynamic DNS updates
- Fixed IP address allocation

Note: Our aim is to describe a working example of a DHCP client/server implementation. This is by no means an exhaustive review of all the features available to DHCP. For a more complete understanding, we recommend reviewing the documentation that is distributed with the DHCP RedHat Package Manager (RPM) packages.

DHCP server configuration

The DHCP server was already installed on our SuSE system, as we had selected the Network Server system configuration option when we originally installed Linux. If the DHCP server (called dhcpd) is not installed on your system, you can find the RPM package that contains the server in the n2 directory of the SLES 7 distribution; for example:

```
/suse/n2/dhcp.rpm
```

The dhcp.rpm file can be installed from the root user ID by using the following command:

```
rpm -ivh /suse/n2/dhcp.rpm
```

Note: There may be directories before /suse in your environment, but the filename will be dhcp.rpm.

The DHCP server must be configured before use. A sample configuration file can be found in:

```
/usr/share/doc/packages/dhcp/dhcpd.conf
```

Copy this file to the /etc/dhcpd.conf file, as this is where the DHCP server expects to find its configuration file. Edit this file for your environment.

Using our test system as an example (see Figure 10 on page 29), we now review the contents of our dhcpd.conf configuration file. Example 18 provides a listing of the file.

Example 18 /etc/dhcpd.conf DHCP server configuration file

```
# Global options

ddns-update-style none;
authoritative;

# Interface parameters for Guest LAN "TSTLAN"
subnet 10.0.3.0 netmask 255.255.255.0 {
       range 10.0.3.25 10.0.3.254;
       default-lease-time 2419200;
       max-lease-time 2419200;
       option subnet-mask 255.255.255.0;
       option domain-name "tstlan.itso.ibm.com";
       option domain-name-servers 10.0.3.20;
}

# Interface parameters for Guest LAN "DEVLAN"
subnet 192.168.0.0 netmask 255.255.255.0 {
       range 192.168.0.10 192.168.0.20;
       default-lease-time 604800;
       max-lease-time 604800;
       option subnet-mask 255.255.255.0;
       option domain-name "devlan.itso.ibm.com";
       option domain-name-servers 192.168.0.1;
}
```

There are many parameters available for configuring the DHCP server. In Table 8, we review the parameters we used in our environment. For a complete list and description of all the parameters available, refer to the dhcpd.conf man page (available on your Linux system by typing: `man dhcpd.conf` from the shell).

Table 8 /etc/dhcpd.conf parameters

Parameter	Description
`ddns-update-style`	The DHCP server has the ability to dynamically update the Domain Name Server (DNS). We have not enabled this function. Instead, we use another mechanism for updating the DNS. For more details, see “Dynamic DNS updates” on page 39.

Parameter	Description
authoritative	When a client requests an IP address, the server must determine if the address being requested is valid for that subnet. If the server decides that the request is invalid, a DHCPNAK message is sent to the client. In order to make such decisions, the server must have a complete, definitive picture of the network. If that is the case, it is safe to use the authoritative parameter.
subnet	The subnet parameter describes a particular subnet within your network. In our example, we have defined two subnets, one for each Guest LAN that we have running.
range	All IP addresses within the range parameter are available for lease to DHCP clients.
default-lease-time	The length of time in seconds that a lease will be granted to a client if the client does not request a specific lease time. The “TSTLAN” has a lease time of four weeks. The lease time of the “DEVLAN” is one week.
max-lease-time	The maximum amount of time in seconds that a lease will be granted.
option subnet-mask	The subnet mask that the client will use with its leased IP address.
option domain-name	The domain name that the client will use.
option domain-name-servers	The IP address of one or more Domain Name Servers (DNS) that the client can use for DNS resolution.

Once we have made our modifications to the /etc/dhcpd.conf file, there are a few more steps we need to follow before starting the DHCP server.

Edit the SuSE configuration file /etc/rc.config (remembering to make a backup of it first). Find the line in the file that says:

```
START_DHCPD="no"
```

Change it to read:

```
START_DHCPD="yes"
```

If you do not see a line with **“**START_DHCPD=**”**, add one. The DHCP server will now start automatically when Linux is booted.

Tip: Whenever you make a change to /etc/rc.config, you must run the SuSE configuration script SuSEconfig in order to apply the changes to the systems’s configuration files.

Edit the DHCP server configuration file /etc/rc.config.d/dhcpd.rc.config (again, remembering to make a backup before editing the file).

We changed the line which read:

```
DHCPD_INTERFACE="eth0"
```

to read:

```
DHCPD_INTERFACE="eth1 eth2"
```

as our DHCP server listens on these two interfaces, each one connected to a different z/VM Guest LAN (see Figure 10 on page 29).

Finally, before starting the server we need to create an empty DHCP lease database file. This file will store all the IP address lease information. The syntax to create an empty lease file is:

```
touch /var/lib/dhcp/dhcpd.leases
```

We are now ready to start our DHCP server. We can either reboot the system and let the server start automatically, or we can start the server manually with the command:

```
rcdhcpd start
```

If your configuration was successful, output similar to that shown in Example 19 should appear in the /var/log/messages file.

Example 19 DHCP server startup messages

```
Nov 27 20:19:37 lnx20 dhcpd: Internet Software Consortium DHCP Server V3.0rc12
Nov 27 20:19:37 lnx20 dhcpd: Copyright 1995-2001 Internet Software Consortium.
Nov 27 20:19:37 lnx20 dhcpd: All rights reserved.
Nov 27 20:19:37 lnx20 dhcpd: For info, please visit http://www.isc.org/products/DHCP
Nov 27 20:19:37 lnx20 dhcpd: Wrote 0 deleted host decls to leases file.
Nov 27 20:19:37 lnx20 dhcpd: Wrote 0 new dynamic host decls to leases file.
Nov 27 20:19:37 lnx20 dhcpd: Wrote 0 leases to leases file.
Nov 27 20:19:37 lnx20 dhcpd: Listening on LPF/eth2/00:04:ac:00:00:00/10.0.3.0/24
Nov 27 20:19:37 lnx20 dhcpd: Sending on  LPF/eth2/00:04:ac:00:00:00/10.0.3.0/24
Nov 27 20:19:37 lnx20 dhcpd: Listening on LPF/eth1/00:04:ac:00:00:01/192.168.0.0/24
Nov 27 20:19:37 lnx20 dhcpd: Sending on  LPF/eth1/00:04:ac:00:00:01/192.168.0.0/24
Nov  8 11:02:13 lnx20 dhcpd: Sending on   Socket/fallback/fallback-net
```

DHCP client configuration

The DHCP client, dhcpcd, was already installed on our system. If this client is not installed on your SuSE system, you can find the RPM package that contains this code in the n1 directory of the SLES 7 distribution; for example:

```
/suse/n1/dhcpcd.rpm
```

If you need to install the dhcpcd.rpm file, this can be done from the root user ID by using the following command:

```
rpm -ivh /suse/n1/dhcpcd.rpm
```

Note: There may be directories before /suse in your environment, but the filename will be dhcpcd.rpm.

Before we can use the DHCP client to automatically lease an IP address at boot time, we need to make some modifications to the system.

Edit the SuSE configuration file /etc/rc.config (remembering to make a backup of it first). Change the line that reads:

```
IFCONFIG_0=...
```

to:

```
IFCONFIG_0="dhcpclient"
```

Tip: Whenever you make a change to /etc/rc.config, you must run the SuSE configuration script, SuSEconfig, in order to apply the changes to the systems's configuration files.

Edit the DHCP client configuration file /etc/rc.config.d/dhcpcd.rc.config (again, remembering to make a backup before editing the file), adding the lines shown in Example 20.

Example 20 Additions to /etc/rc.config.d/dhcpcd.rc.config

```
DHCLIENT_BROADCAST="yes"
DHCLIENT_DDNS_HOSTNAME_UPDATE="yes"
DHCLIENT_IDENTIFIER="yes"
```

Finally, edit the DHCP client script /etc/init.d/dhclient (make a backup before editing the file).

In the default version of this script, **dhcpcd** is started without the **-B** operand needed to request broadcast responses from the DHCP server. In Example 21, we show modifications to the script to add the broadcast option, as well as hostname (using the **-h** operand) and client identifier (using the **-I** operand) options.

Example 21 /etc/init.d/dhclient

```
test "$DHCLIENT_DEBUG" = "yes" && \
                                    DHCLIENT_ARGS="$DHCLIENT_ARGS -d"
# Add the following line after the line shown above (which should have already existed)
test "$DHCLIENT_BROADCAST" = "yes" && \
                                    DHCLIENT_ARGS="$DHCLIENT_ARGS -B"
test "$DHCLIENT_DDNS_HOSTNAME_UPDATE" = "yes" && \
                                    DHCLIENT_ARGS="$DHCLIENT_ARGS -h $HOSTNAME"
test "$DHCLIENT_IDENTIFIER" = "yes" && \
                                    DHCLIENT_ARGS="$DHCLIENT_ARGS -I $HOSTNAME"
```

Now that we have made these changes, we can either reboot the Linux guest and automatically lease an IP address as part of the boot process, or we could manually test our DHCP client by typing:

```
dhcpcd -B eth0 -h lnx17 -I lnx17
```

We chose to reboot our Linux DHCP client machine. In Example 22, we see the output from the DHCP server's /var/log/messages file.

Example 22 Output of /var/log/messages for a successful IP address lease

```
Nov 27 20:27:09 lnx20 dhcpd: DHCPDISCOVER from 00:04:ac:00:00:02 (lnx17) via eth2
Nov 27 20:27:09 lnx20 dhcpd: DHCPOFFER on 10.0.3.100 to 00:04:ac:00:00:02 (lnx17) via eth2
Nov 27 20:27:09 lnx20 kernel:  qeth: QETH_IP_VERSION is 0
Nov 27 20:27:09 lnx20 kernel:  qeth: skb->protocol=x0=0
Nov 27 20:27:09 lnx20 kernel:  qeth: skb:ff ff ff ff  ff ff 00 04  ac 00 00 00  08 00 45 10
Nov 27 20:27:09 lnx20 kernel:  qeth: skb:01 48 00 00  00 00 10 11  9c 82 0a 00  03 14 ff ff
Nov 27 20:27:09 lnx20 dhcpd: DHCPREQUEST for 10.0.3.100 (10.0.3.20) from 00:04:ac:00:00:02
(lnx17) via eth2
```

```
Nov 27 20:27:09 lnx20 dhcpd: DHCPACK on 10.0.3.100 to 00:04:ac:00:00:02 (lnx17) via eth2
Nov 27 20:27:09 lnx20 kernel:  qeth: QETH_IP_VERSION is 0
Nov 27 20:27:09 lnx20 kernel:  qeth: skb->protocol=x0=0
Nov 27 20:27:09 lnx20 kernel:  qeth: skb:ff ff ff ff  ff ff 00 04  ac 00 00 00  08 00 45 10
Nov 27 20:27:09 lnx20 kernel:  qeth: skb:01 48 00 00  00 00 10 11  9c 82 0a 00  03 14 ff ff
```

Dynamic DNS updates

In a DHCP environment, the Linux client will lease its IP address from a DHCP server. We can then use that IP address to connect to the client machine.

However, we still cannot access the machine using its hostname. As most of the Linux machines running in the z/VM Guest LAN will probably be servers of one kind or another, it would be useful if we could address those machines by hostname and not just by IP address.

To do this, we need a mechanism to dynamically update the LAN's Domain Name Server (DNS) to provide IP address and hostname information for our guest machine.

Note: The Internet Software Consortium's (ISC) DHCP server has a Dynamic DNS update feature. This feature has two update modes (in version 3.0rc12) called ad-hoc and interim. The modes are controlled by the parameter `ddns-update-style` in the /etc/dhcpd.conf file.

The ad-hoc mode has been deprecated in later releases of the DHCP software and the ISC documentation says that it does not work. The interim mode differs slightly from the draft standards that are currently being reviewed by the Internet Engineering Task Force (IETF).

During testing, we used the interim mode. Given that the Dynamic DNS updating protocols are still in a state of flux, we also wrote and tested our own alternative method of dynamically updating the DNS, which does not have to rely on the DHCP server.

We present both of these methods in the following sections. Use one or the other for your environment—do not implement both methods.

Allowing updates to the DNS

Before a machine can update a DNS server's records it must be granted the authority to do so. An extract from our /etc/named.conf DNS configuration file is shown in Example 23. We have added an `allow-update` parameter for each of the two zones listed.

Example 23 /etc/named.conf

```
zone "tstlan.itso.ibm.com" {
        type master;
        file "pz/tstlan.itso.ibm.com";
allow-update { 10.0.3.0/24; };

};

zone "3.0.10.in-addr.arpa" {
        type master;
        file "pz/10.0.3";
allow-update { 10.0.3.0/24; };
};
```

Note: We can secure the DNS update process by implementing the Secret Key Transaction Authentication for DNS (TSIG) protocol. See RFC 2845, which is available from:

http://www.ietf.org/rfc.html

However, for simplicity and ease of understanding, we have chosen not to implement TSIG in our examples. Also, in a trusted z/VM environment it may not be necessary to add this additional infrastructure.

Dynamic DNS using the DHCP server

In order to use the DHCP server's dynamic DNS update facility, we must add two additional global parameters to the /etc/dhcpd.conf file. An example is shown in Example 24.

Example 24 /etc/dhcpd.conf file

```
ddns-update-style interim;
allow client-updates;
```

Example 25 shows the output of the DHCP server's syslog. From the example, we can see that as the client leases an IP address, the DHCP server updates the DNS with details about that client machine.

Example 25 /var/log/messages output showing dynamic DNS update

```
Nov 27 20:45:23 lnx20 dhcpd: DHCPDISCOVER from 00:04:ac:00:00:01 (lnx23) via eth2
Nov 27 20:45:23 lnx20 dhcpd: DHCPOFFER on 10.0.3.99 to 00:04:ac:00:00:01 (lnx23) via eth2
Nov 27 20:45:23 lnx20 kernel:  qeth: QETH_IP_VERSION is 0
Nov 27 20:45:23 lnx20 kernel:  qeth: skb->protocol=x0=0
Nov 27 20:45:23 lnx20 kernel:  qeth: skb:ff ff ff ff  ff ff 00 04  ac 00 00 00  08 00 45 10
Nov 27 20:45:23 lnx20 kernel:  qeth: skb:01 48 00 00  00 00 10 11  9c 82 0a 00  03 14 ff ff
Nov 27 20:45:23 lnx20 dhcpd: if IN A lnx23.tstlan.itso.ibm.com domain doesn't exist add
1209600 IN A lnx23.tstlan.itso.ibm.com 10.0.
Nov 27 20:45:23 lnx20 dhcpd: delete IN PTR 99.3.0.10.in-addr.arpa. add 1209600 IN PTR
99.3.0.10.in-addr.arpa. lnx23.tstlan.itso.ibm.
Nov 27 20:45:23 lnx20 dhcpd: DHCPREQUEST for 10.0.3.99 (10.0.3.20) from 00:04:ac:00:00:01
(lnx23) via eth2
Nov 27 20:45:23 lnx20 dhcpd: DHCPACK on 10.0.3.99 to 00:04:ac:00:00:01 (lnx23) via eth2
Nov 27 20:45:23 lnx20 kernel:  qeth: QETH_IP_VERSION is 0
Nov 27 20:45:23 lnx20 kernel:  qeth: skb->protocol=x0=0
Nov 27 20:45:23 lnx20 kernel:  qeth: skb:ff ff ff ff  ff ff 00 04  ac 00 00 00  08 00 45 10
Nov 27 20:45:23 lnx20 kernel:  qeth: skb:01 48 00 00  00 00 10 11  9c 82 0a 00  03 14 ff ff
```

Dynamic DNS - alternative method

This method does not use the DHCP server to update the DNS, but instead uses an additional script in the Linux startup process so that a machine can dynamically update the DNS once it has leased an IP address. We created a script called /etc/init.d/dhcp_nsupdate. A listing of the script is shown in Example 26. Make certain that you set this script to be executable by the file owner (root).

Example 26 /etc/init.d/dhcp_nsupdate script

```
#!/bin/sh
#
# /sbin/init.d/dhcp_nsupdate
# Description:    Dynamic update to DNS server after lease of IP address
# Determine the base and follow a runlevel link name.
```

```
base=${0##*/}
link=${base#*[SK][0-9][0-9]}


IPADDRESS=`ifconfig  | grep "inet addr:" | grep -v "127.0.0.1" | \
        awk -F: {'print $2'} | cut -d\  -f 1`

DOMAINNAME=`/bin/domainname`

echo "update add $HOSTNAME.$DOMAINNAME 20000 A $IPADDRESS" > /var/named/dhcp_update_command
echo " " >> /var/named/dhcp_update_command

case "$1" in
    "start")
        echo "Our IP address is:" $IPADDRESS
        echo "Our HOSTNAME is:" $HOSTNAME
        echo "Our DOMAINNAME is:" $DOMAINNAME
        echo "Adding DNS entry for this server"
        nsupdate /var/named/dhcp_update_command
        rcinetd restart
     ;;
    "stop")
        echo "Removing DNS entry for this server"
        rm /var/named/dhcp_update_command
        echo "update delete $HOSTNAME.$DOMAINNAME A" > /var/named/dhcp_update_command
        echo " " >> /var/named/dhcp_update_command
        nsupdate /var/named/dhcp_update_command
     ;;
         *)
        echo "Usage: $0 {start|stop}"
exit 1
;;
esac
```

From the listing, we can see that the script determines the IP address, hostname, and domain name of the Linux guest, and then runs the **nsupdate** command to update the DNS server. On machine startup, the script will add a DNS record; on shutdown, it will remove the DNS record.

In order to run this script, either automatically at Linux startup and shutdown or manually from the shell, we need to create some symbolic links in the filesystem. An example of the commands used to set up these links is shown in Example 27.

Example 27 Setting up the symbolic links

```
cd /sbin
ln -s /etc/init.d/dhcp_nsupdate rcdhcp_nsupdate

cd /etc/rc.d/rc3.d/
ln -s /etc/init.d/dhcp_nsupdate S24dhcp_nsupdate
ln -s /etc/init.d/dhcp_nsupdate K09dhcp_nsupdate
```

Now that we have made these additional changes, this alternative method of DNS updating is ready to be tested. Either shut down your Linux guest and reboot it, or use the **rcdhcp_nsupdate start** command from the Linux shell.

Tip: A runlevel defines which scripts are to be run when Linux is booted and shut down. The default runlevel for the SLES 7 Linux system is runlevel 3. We have defined symbolic links so that the **dhcp_nsupdate** script will run whenever Linux is booted or shut down in runlevel 3.

A naming convention is applied to the symbolic links. The first character is either an S or a K, for startup or shutdown (kill), respectively. A sequence number follows the first character. This number dictates in which order startup or shutdown scripts are to run. This is useful for managing dependencies between processes running in the Linux system. For example, we want **S24dhcp_nsupdate** to run after **S05network**, which is the main network startup script.

You can choose your own numbering to fit in with your standards. The dependency for adding entries to the DNS server is that we must have first started our networking devices and leased an IP address. The dependency for removing entries from the DNS is to run the **dhcp_nsupdate** script before dropping network connectivity.

Fixed IP address allocation

There may be certain Linux machines in your Guest LAN that require a fixed IP address due to specific application or operational requirements. We could define these Linux guests with static IP addresses outside of DHCP control. However, that means that we would still need to configure IP addressing information for each of these machines, which is something we are trying to avoid.

An easier approach would be to let the DHCP server handle the IP address allocation. In that scenario, the DHCP client presents the server with a unique identifier as part of its request for an IP address. The server uses that identifier to determine if the client is to receive a fixed IP address instead of an address from a pool. The unique identifier can be a machine's MAC address or a user-specified *dhcp-client-identifier*.

Restriction: In a z/VM Guest LAN environment, we cannot guarantee that a guest machine will have the same MAC address every time it logs on. Therefore, we cannot use MAC addresses as a unique identifier for fixed IP address allocation purposes.

In "DHCP client configuration" on page 37, we configured the DHCP client software to send a unique identifier to the server. In our example, we used:

```
dhcpcd -B eth0 -h lnx17 -I lnx17
```

The `-I` parameter is the dhcp-client-identifier.

Note: We chose to use the linux hostname as the unique dhcp-client-identifier. As we had complete control over the system, that was a safe thing to do.

However, if you do not have complete control over your environment, you should develop a strategy so that the dhcp-client-identifier is, in fact, unique. Otherwise, it is possible that a server may be given an IP address meant for another machine on your system.

On the server side, we define which systems are to have fixed IP addresses by updating the /etc/dhcpd.conf file.

In Example 28, two machines are configured to have fixed IP addresses.

Example 28 Extract of /etc/dhcpd.conf showing fixed IP address allocations

```
host lnx17.tstlan.itso.ibm.com {
     option dhcp-client-identifier 00:6c:6e:78:31:37:00;
     fixed-address 10.0.3.99;
}

host lnx23.tstlan.itso.ibm.com {
     option dhcp-client-identifier 00:6c:6e:78:32:33:00;
     fixed-address 10.0.3.98;
}
```

Important: You may be wondering why a dhcp-client-identifier of lnx17 on the client side is defined as 00:6c:6e:78:31:37:00 in Example 28. In testing, we tried to use the string “lnx17” as the dhcp-client-identifier, but that didn’t work.

After reading the source code and packet-tracing the conversation, we determined that the DHCP server was actually expecting to see a 6-byte MAC address (in the usual hexadecimal format) with a 1-byte interface type in front of the address.

Armed with that knowledge. we converted our “lnx17” string to ASCII hexadecimal. The result was 6c:6e:78:31:37. As we only had 5 bytes, we added 00 to the end of the identifier. We also added 00 to the front of the identifier for the interface type. How did we know to use 00’for the interface type? We got that information from the packet trace.

We tested this method up to an 8-byte guest name (the maximum allowed by CP); for example, “linux678”. In that situation, our /etc/dhcpd.conf definition looked as follows:

```
host linux678.tstlan.itso.ibm.com {
     option dhcp-client-identifier 00:6c:69:6e:75:78:36:37:38:00;
     fixed-address 10.0.3.77;
}
```

In ASCII, hexadecimal “linux678” is 6c:69:6e:75:78:36:37:38 and again we added a 00 to both the beginning and the end of the identifier.

Linux installation using DHCP

With a DHCP server in a z/VM Guest LAN, we now theoretically have the option to dynamically lease an IP address when installing a Linux instance, instead of the previous requirement of having to hardcode a specific IP address. However, as no Linux distributions at time of writing (November 2002) were at the requisite kernel and qeth device driver levels, this capability is not yet available. We anticipate that once the Linux for zSeries and S/390 distributions are at the correct software levels, Linux installation in a Guest LAN will be able to take advantage of DHCP.

Summary

This Redpaper has introduced the concepts of IP broadcast and DHCP, and has outlined a method of implementing a DHCP environment in a z/VM Guest LAN. We believe that the introduction of broadcast support, and through it DHCP to a z/VM Guest LAN, is very beneficial to systems programmers.

Instead of hardcoding network information in each guest, we can now obtain the information dynamically from a DHCP server. This reduces the time and effort required to install, clone, or administer a Linux guest machine. This should prove particularly useful in large-scale Linux deployments.

Appendix

Upgrading your Linux kernel

Attention: Upgrading the Linux kernel is a straightforward process if you follow the instructions accurately and are confident that you know what you are doing. We recommend that, until you are comfortable with upgrading kernels, you only practice on machines which are expendable and can be rebuilt if you make a mistake.

If you have a support contract for Linux on zSeries or System/390, upgrading your Linux kernel to 2.4.19 may affect your support contract. We anticipate that standard distributions based on the 2.4.19 (or later) kernel will become available in late 2002.

For additional information, see Appendix D, “Building the kernel” in the publication *Linux for zSeries and S/390 Device Drivers and Installation Commands, July 31, 2002, Linux Kernel 2.4* LNUX-1303-01, available at:

`http://oss.software.ibm.com/linux390/docu/lnuxdd01.pdf`

As previously stated, the examples in this Redpaper have been based on the 31-bit SuSE Linux Enterprise Server (SLES) 7 distribution. While we did not have the resources to test other distributions, any system that runs a kernel at 2.4.19 or higher with the appropriate QETH device driver should be able to exploit broadcast support and thus enjoy the benefits of DHCP.

Your build system must have the following levels of software installed (as a minimum):

- gcc version 2.95.3 or later supporting Linux for zSeries and S/390
- binutils version 2.11.90.0.27 or later supporting Linux for zSeries and S/390
- glibc 2.2.4 or later supporting Linux for zSeries and S/390
- sed
- bash
- make version 3.77 or later

Follow these steps to upgrade from the 2.4.7 kernel to the 2.4.19 kernel:

1. Obtain the “vanilla” 2.4.19 Linux kernel.
2. Obtain the Linux for zSeries and S/390-specific patches and OCO modules.
3. Patch the kernel with the zSeries and S/390-specific patches.
4. Configure and build the kernel.
5. Copy OCO modules.
6. Copy the new kernel files, run zipl and re-IPL.
7. Build s390-tools.

Obtain the “vanilla” 2.4.19 Linux kernel

We downloaded the “vanilla” (standard) 2.4.19 Linux kernel from the Linux kernel ftp server:

```
ftp://ftp.kernel.org/pub/linux/kernel/v2.4
```

The file was called:

```
linux-2.4.19.tar.gz
```

Obtain the Linux for zSeries and S/390-specific patches and OCO modules

We downloaded the zSeries and S/390-specific patches from the IBM developerworks Web site:

```
http://oss.software.ibm.com/linux390/current2_4_19-may2002.shtml
```

The files downloaded to the /usr/src/kernel directory were:

- linux-2.4.19-s390-may2002.tar.gz
- linux-2.4.19-s390-may2002-1.tar.gz
- linux-2.4.19-s390-may2002-2.tar.gz
- linux-2.4.19-timer-1-may2002.tar.gz
- linux-2.4.19-s390-kerntypes-may2002.tar.gz

We downloaded the s390-tools package:

s390-tools-1.1.3-may2002.tar.gz

to the /usr/src/s390-tools directory.

We also downloaded the version of the OSA-Express QETH driver Object Code Only (OCO) module that provides support for IP broadcast from:

```
http://www10.software.ibm.com/developerworks/opensource/linux390/special_oco_su_2.4.shtml
```

We downloaded the qeth-susekernel-2.4.19-s390-1.tar.gz file to the /usr/src/OCOmodule directory.

Note: In earlier levels of the Linux for zSeries and S/390 kernel, we had to download both a QDIO and QETH OCO module. Starting with the 2.4.19 kernel level, the QDIO module is now provided in source-code as part of the kernel patch.

Patch the kernel

The Linux kernel source resides in /usr/src/linux directory. You may already have a copy of an earlier Linux kernel at this location. If you want to keep it, we recommend you rename this directory.

Unwind the vanilla Linux kernel and patch it with the zSeries-specific patches as shown in Example 29.

Example 29 Patching the kernel

```
# cd /usr/src
# tar -zxvf linux-2.4.19.tar.gz
# ln -s linux-2.4.19 linux
# cd kernel
# tar -zxf linux-2.4.19-s390-may2002.tar.gz
# tar -zxf linux-2.4.19-s390-may2002-1.tar.gz
```

```
# tar -zxf linux-2.4.19-s390-may2002-2.tar.gz
# tar -zxf linux-2.4.19-timer-1-may2002.tar.gz
# tar -zxf linux-2.4.19-s390-kerntypes-may2002.tar.gz
# cd ../linux
# patch -p1 < ../kernel/linux-2.4.19-s390-may2002.diff
# patch -p1 < ../kernel/linux-2.4.19-s390-1-may2002.diff
# patch -p1 < ../kernel/linux-2.4.19-s390-2-may2002.diff
# patch -p1 < ../kernel/linux-2.4.19-timer-1-may2002.tar.gz
# patch -p1 < ../kernel/linux-2.4.19-s390-kerntypes-may2002.diff
```

Then clean up the distribution by running the command **`make distclean`** from the /usr/src/linux directory. This will reset options and remove any old object files.

Configure and build the kernel

From the /usr/src/linux directory, type: `make menuconfig` to invoke the kernel configuration utility. You will be presented with a screen similar to that shown in Example 30. The areas you need to review in relation to this Redpaper are denoted by the **bold** text. Note, however, that there may be additional options that you want to set for your environment.

Example 30 make menuconfig - main screen

```
Code maturity level options  --->
Loadable module support  --->
Processor type and features  --->
General setup  --->
SCSI support  --->
Block device drivers  --->
Multi-device support (RAID and LVM)  --->
Character device drivers  --->
Network device drivers  --->
Miscellaneous  --->
Networking options  --->
File systems  --->
Kernel hacking  --->
---
Load an Alternate Configuration File
Save Configuration to an Alternate File
```

The Processor type and features screen has IEEE FPU emulation, which can be turned off for G5 or later machines which have IEEE FPU instructions in the hardware, as shown in Example 31.

Example 31 Processor type and features

```
[*] Symmetric multi-processing support
[ ] IEEE FPU emulation
```

The General setup screen has the No HZ timer ticks in idle setting which should be turned on, as shown in Example 32. This is especially important when many Linux images are running under z/VM.

Example 32 General setup

```
[*] Fast IRQ handling
[*] Process warning machine checks
[*] Use chscs for Common I/O
<M> QDIO support
[ ]    Performance statistics in /proc
[*] Builtin IPL record support
(vm_reader) IPL method generated into head.S
[*] Networking support
```

```
[*] System V IPC
[ ] BSD Process Accounting
[*] Sysctl support
<*> Kernel support for ELF binaries
< > Kernel support for MISC binaries
[ ] Show crashed user process info
[*] Pseudo page fault support
[ ] VM shared kernel support
[*] No HZ timer ticks in idle
[ ]   Idle HZ timer on by default
```

In the networking options screen shown in Example 33, we need to make sure that the Packet socket option and the Socket Filtering option are enabled, as these are required for the DHCP server.

Example 33 Networking options

```
<*> Packet socket
[ ]   Packet socket: mmapped IO
< > Netlink device emulation
[*] Network packet filtering (replaces ipchains)
[ ]   Network packet filtering debugging
[*] Socket Filtering
<*> Unix domain sockets
[*] TCP/IP networking
[*]   IP: multicasting
[ ]   IP: advanced router
[ ]   IP: kernel level autoconfiguration
< >   IP: tunneling
< >   IP: GRE tunnels over IP
[ ]   IP: multicast routing
[ ]   IP: ARP daemon support (EXPERIMENTAL)
[ ]   IP: TCP Explicit Congestion Notification support
[ ]   IP: TCP syncookie support (disabled per default)
IP: Netfilter Configuration  --->
<M>   The IPv6 protocol (EXPERIMENTAL)
IPv6: Netfilter Configuration  --->
[*]   Prepare net_device struct for shared IPv6 cards
< >   Kernel httpd acceleration (EXPERIMENTAL)
[ ] Asynchronous Transfer Mode (ATM) (EXPERIMENTAL)
<M> 802.1Q VLAN Support (EXPERIMENTAL)
---
```

In the File systems menu shown in Example 34, be sure to turn off “/dev file system support” unless you specifically want to use it and you have properly configured your system to use DEVFS.

Make sure that you have “Second extended fs support” enabled; this is for the ext2 filesystem. You may be interested in turning on support for the ext3 filesystem, selected via the option “Ext3 journalling file system support”.

Example 34 File systems menu

```
[ ] Quota support
< > Kernel automounter support
< > Kernel automounter version 4 support (also supports v3)
< > Reiserfs support
< > ADFS file system support
< > Amiga FFS file system support (EXPERIMENTAL)
< > Apple Macintosh file system support (EXPERIMENTAL)
```

```
< > BFS file system support (EXPERIMENTAL)
< > Ext3 journalling file system support
< > DOS FAT fs support
< > EFS file system support (read only) (EXPERIMENTAL)
< > Journalling Flash File System (JFFS) support
< > Journalling Flash File System v2 (JFFS2) support
< > Compressed ROM file system support
[ ] Virtual memory file system support (former shm fs)
< > ISO 9660 CDROM file system support
< > Minix fs support
< > FreeVxFS file system support (VERITAS VxFS(TM) compatible)
< > NTFS file system support (read only)
< > OS/2 HPFS file system support
[*] /proc file system support
[ ] /dev file system support (EXPERIMENTAL)
[ ] /dev/pts file system for Unix98 PTYs
< > QNX4 file system support (read only)
< > ROM file system support
<*> Second extended fs support
< > System V/Xenix/V7/Coherent file system support
< > UDF file system support (read only)
< > UFS file system support (read only)
Network File Systems  --->
Partition Types  --->
```

Once you have made these selections, then exit from the menuconfig utility, remembering to save the configuration as you exit.

You are now ready to build the kernel using the commands illustrated in Example 35.

Example 35 Commands to build the kernel

```
# make dep
...
# make image
...
# make modules
...
# make modules_install
...
```

Copy OCO modules

Copy the downloaded QETH OCO module to the correct directory, as shown in Example 36.

Example 36 Copy QETH module

```
# cd /usr/src/OCOmodule
# tar -zxf qeth-2.4.19-s390-1-may2002.tar.gz
# cp qeth-2.4.19-s390-1-may2002.o /lib/modules/2.4.19/kernel/net/qeth.o
```

Copy the new kernel files, run zipl and re-IPL

Copy the new kernel, system map, and kernel configuration files to the correct directories, as shown in Example 37.

Example 37 Copy the kernel files

```
# cd /boot/kernel
# cp /usr/src/linux/arch/s390/boot/image image-2.4.19
# cp /usr/src/linux/System.map System.map-2.4.19
# cp /usr/src/linux/.config .config-2.4.19
```

Before we can re-IPL our Linux guest, we must prepare the boot disk to load our newly built 2.4.19 kernel.

To do this, we run the zSeries Initial Program Loader (zIPL) utility. An example of running this utility is shown in Example 38.

Example 38 Running zipl

```
# zipl --target=/boot/zipl --image=/boot/kernel/image-2.4.19 --parmfile=/boot/parmfile

building bootmap    : /boot/zipl/bootmap
adding Kernel Image : /boot/kernel/image-2.4.19 located at 0x00010000
adding Parmline     : /boot/parmfile located at 0x00001000
Bootloader for ECKD type devices with classic layout installed.
Syncing disks....
...done
```

We can now re-IPL our Linux guest. Once the system has started, run the **uname -a** command to determine if the system is now at the 2.4.19 level of Linux kernel, as seen in Example 39.

Example 39 Running uname

```
# uname -a

Linux lin005 2.4.19 #1 SMP Wed Nov 13 22:10:53 EST 2002 s390 unknown
```

Build s390-tools

As the name implies, the “s390-tools” package contains a set of zSeries and S/390-specific utilities. The utilities include:

- **dasdfmt**

  Used to low-level format ECKD DASD with either the classic Linux disk layout or the new z/OS-compatible disk layout.

- **fdasd**

  Used to create or modify partitions on ECKD DASD formatted with the z/OS-compatible disk layout.

- **dasdview**

  Used to display DASD and VTOC information or to dump the contents of a DASD to the console.

- **zipl**

  Used to make either a DASD or a tape bootable for system IPL or system dump.

- **zgetdump**

  Used to retrieve system dumps from either tape or DASD.

- **qetharp**

  A tool to read and flush the ARP cache on the OSA Express Direct network cards.

- **tape390_display**

  Used to display messages on the display unit of a zSeries tape device.

- **osasnmpd**

  A ucd-snmp subagent implementing MIBs provided by OSA-Express Fast Ethernet, Gigabit Ethernet, High Speed Token Ring and ATM Ethernet LAN Emulation in QDIO mode.

We need to build a new version of this package that will be fully compatible with the 2.4.19 kernel.

Note: The zipl utility can be run purely from the command line or with a zipl configuration file, typically /etc/zipl.conf. In order for zipl to read the configuration file, we need to compile the zipl source code with an additional parser package available from the Internet. The package is called parsecfg-3.6.7 and it can be downloaded from:

http://www.netsw.org/system/libs/config

To build s390-tools, perform the following steps illustrated in Example 40 (note that we made a backup of the current zipl program).

Example 40 Build s390-tools package

```
# cp /sbin/zipl /sbin/zipl.orig
# cd /usr/src/s390-tools
# tar -zxf s390-tools-1.1.3-may2002.tar.gz
# ln -s s390-tools-1.1.3 s390-tools
# tar -zxf parsecfg-3.6.7.tar.gz
# ln -s parsecfg-3.6.7 parsecfg
# cd s390-tools
# cp ../parsecfg/src/parsecfg.h zipl/include
# cp ../parsecfg/src/parsecfg.c zipl/src
# make
# make install
# make clean
```

We now have a new “s390-tools” suite of utilities, compatible with the Linux 2.4.19 kernel.

Related publications

IBM Redbooks

- *Building Linux Systems Under IBM VM*, REDP0120

  http://www.ibm.com/redbooks/abstracts/redp0120.html
- *Linux on IBM @server zSeries and S/390: ISP/ASP Solutions*, SG24-6299

  http://www.ibm.com/redbooks/abstracts/sg246299.html
- *Linux on IBM @server zSeries and S/390: Distributions*, SG24-6264

  http://www.ibm.com/redbooks/abstracts/sg246264.html
- *Linux on IBM @server zSeries and S/390: Large Scale Linux Deployment*, SG24-6824

  http://www.ibm.com/redbooks/abstracts/sg246824.html
- *zSeries HiperSockets*, SG24-6816

  http://www.ibm.com/redbooks/abstracts/sg246816.html

Other resources

- *z/VM CP Planning and Administration, Version 4 Release 3.0*, SC24-6043
- *z/VM CP Command and Utility Reference, Version 4 Release 3.0*, SC24-6008
- *The DHCP Handbook: Understanding, Deploying, and Managing Automated Configuration Services*, by Ralph Droms, Ph.D. and Ted Lemon. Macmillan Technical Publishing, 1999. ISBN 1578701376.

Referenced Web sites

- Linux for zSeries Device Driver code and documentation

  http://oss.software.ibm.com/developerworks/opensource/linux390/current2_4_17-may2002.shtml
  http://www10.software.ibm.com/developerworks/opensource/linux390/docu/lnuxdd01.pdf
- Internet Engineering Task Force standards

  http://www.ietf.org/html.charters/dhc-charter.html
  http://www.ietf.org/rfc.html
- DHCP server

  http://www.isc.org
- DHCP client

  http://www.phystech.com/download/

About the author

Simon Williams is a Senior I/T Specialist with IBM Australia. He likes to specialize in zSeries “new technologies” such as Linux and TCP/IP networking. However, he can still occasionally be spotted writing a bit of assembler or JCL. His technical publications include ITSO Redbooks on Domino/390, Linux for zSeries ISP/ASP solutions, and a Redpaper on how to install Linux guests under VM. Before joining IBM, he was an MVS Systems Programmer and has been working with mainframe systems since 1988.

Acknowledgements

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

Roy Costa, Greg Geiselhart, Mike MacIsaac, and Dennis Musselwhite.

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 Internet note to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

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

IBM eServer™	OS/2®	System/390®
DFS™	Perform™	z/OS™
ECKD™	Redbooks(logo)™ 	z/VM™
IBM®	Redbooks™	zSeries™
Multiprise®	S/390®	
MVS™	SP™	

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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