Vic Cross

Linux on IBM eserver zSeries and S/390: Virtual Router Redundancy Protocol on VM Guest LANs

Overview

This Redpaper describes the Virtual Router Redundancy Protocol (VRRP) and its use in a Linux VM Guest LAN environment. The following topics are covered:

- About VRRP
- Linux VRRP implementations
- Building and installing keepalived
- VRRP on Guest LANs using keepalived
- VRRP details

About VRRP

When building networks, one of the ways to increase availability is to provide redundancy for critical components. This usually involves duplicating routers, switches and links to ensure continuity of service across failures. Dynamic routing protocols are used to keep the network running, routing traffic around network problems.

One place where it can be difficult to provide this level of redundancy is at the endpoints of the network. There are two main reasons for this:

- It is often impractical to provide multiple network connections for end-stations (particularly desktop workstations) due to prohibitive cost and duplication of horizontal cabling.
- Running dynamic routing protocols on end-stations, to allow them to take advantage of multiple network paths and/or multiple gateways, is not feasible due to the network overhead and resulting complexity of the routing environment.

The Virtual Router Redundancy Protocol (VRRP) is an Internet standard described in RFC2338. It gives network designers a way to provide reliable, redundant gateway service for IP end-stations.

VRRP introduces the concept of a “virtual router” that is addressed by IP clients requiring gateway service. The actual routing service is provided by physical routers running the VRRP protocol. An example of this is shown in Figure 1.

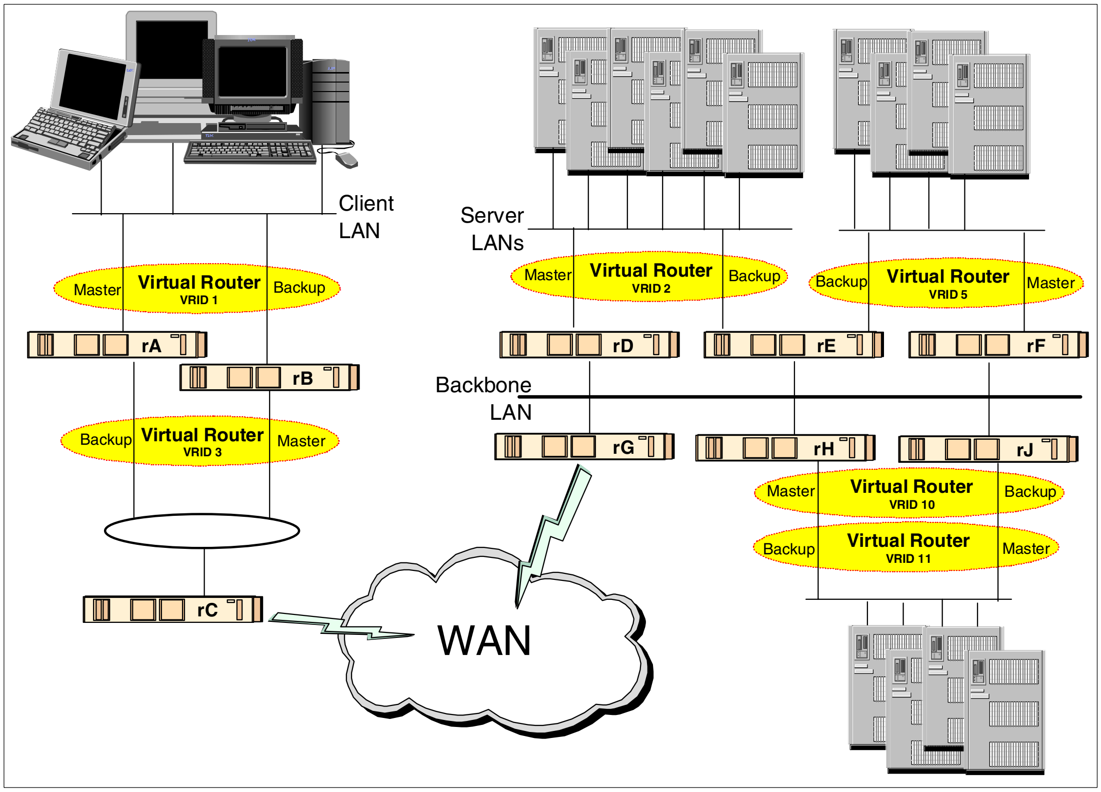


Figure 1 VRRP concepts

This illustration shows a number of VRRP concepts.

- Router rA is the master of virtual router VRID 1, and the backup for VRID 3. At this time, it handles the routing of packets addressed to the VIP for VRID1, and is ready to take on the routing role for VRID 3.
- Router rB is the master of virtual router VRID 3, and the backup for VRID 1. At this time, it handles the routing of packets addressed to the VIP for VRID3, and is ready to take on the routing role for VRID 1.
- Router rC does not have VRRP function, but uses the VIP for VRID 3 to reach the Client LAN subnet.
- Router rD is the master of VRID 2. Router rF is the master of VRID 5. Router rE is the backup for both of these VRIDs. If rD or rF fails, rE will become the master for that VRID. In fact, both rD and rF could fail at the same time; the fact that a VRRP router is a master for one VRID does not preclude it from being master for another.
- Router rG is the WAN gateway for the Backbone LAN. All of the routers attached to the backbone are sharing routing information with the routers on the WAN using a dynamic routing protocol such as OSPF. VRRP is not involved in this, although Router rC will advertise that the path to the Client LAN subnet is via the VIP of VRID 3.

- Router rH is the master of VRID 10, and backup for VRID 11. Likewise, router rJ is the master for VRID 11 and the backup for VRID 10. This is a VRRP load-sharing configuration, and it illustrates that multiple VRIDs can exist on a single router interface.

VRRP can be used as part of a network design that provides almost total routing redundancy for all systems in the network.

VRRP terminology

A number of descriptive terms are introduced by VRRP:

Virtual Router A single router image created through the operation of one or more routers running VRRP.

VRRP Instance A program, implementing VRRP, running on a router. A single VRRP instance can provide VRRP capability for more than one virtual router.

Virtual Router ID Also called VRID, this is a numerical identification of a particular virtual router. VRIDs must be unique on a given network segment.

Virtual Router IP An IP address associated with a VRID that other hosts can use to obtain network service from. The VRIP is managed by the VRRP instances belonging to a VRID.

Note: In this paper, we use the term VRIP to refer to the IP address managed by the VRRP virtual router. Other VRRP documentation may use the term “virtual IP address”, but we will not use that here because of possible confusion with the VIPA function provided by z/OS™ and z/VM™ TCP/IP stacks. VIPA is not related to VRRP in any way.

Virtual MAC address For media that use MAC addressing (such as Ethernet), VRRP instances use a predefined MAC address for all VRRP actions instead of the real adapter MAC address(es). This isolates the operation of the virtual router from the real router providing the routing function. The VMAC is derived from the VRID.

Master The one VRRP instance that performs the routing function for the virtual router at a given time. Only one master is active at a time for a given VRID. Also refers to the state of the VRRP FSM when the VRRP instance is operating as master (that is, “master state”).

Backup VRRP instances for a VRID that are active but not in the master state. Any number of backups can exist for a VRID. Backups are ready to take on the role of master if the current master fails. Also refers to the state of the VRRP FSM when the VRRP instance is operating as backup (that is, “backup state”).

Priority Different VRRP instances are assigned a priority value, as a way of determining which router will take on the role of master if the current master fails. *Priorit* is a number from 1 to 254 (0 and 255 are reserved). The larger the number, the higher the priority.

Owner If the virtual IP address is the same as any of the IP addresses configured on an interface of a router, that router is the owner of the virtual IP address. The priority of the VRRP instance when it is the VIP owner is 255, the highest (and reserved) value.

VRRP in action

Let’s analyze a simple scenario describing how VRRP operates. In this example, we will illustrate how VRRP operates on Ethernet.

Figure 2 shows a simple VRRP configuration, with two routers connecting to a network cloud. We will use VRRP to provide a resilient routing function for the client machines in the LAN.

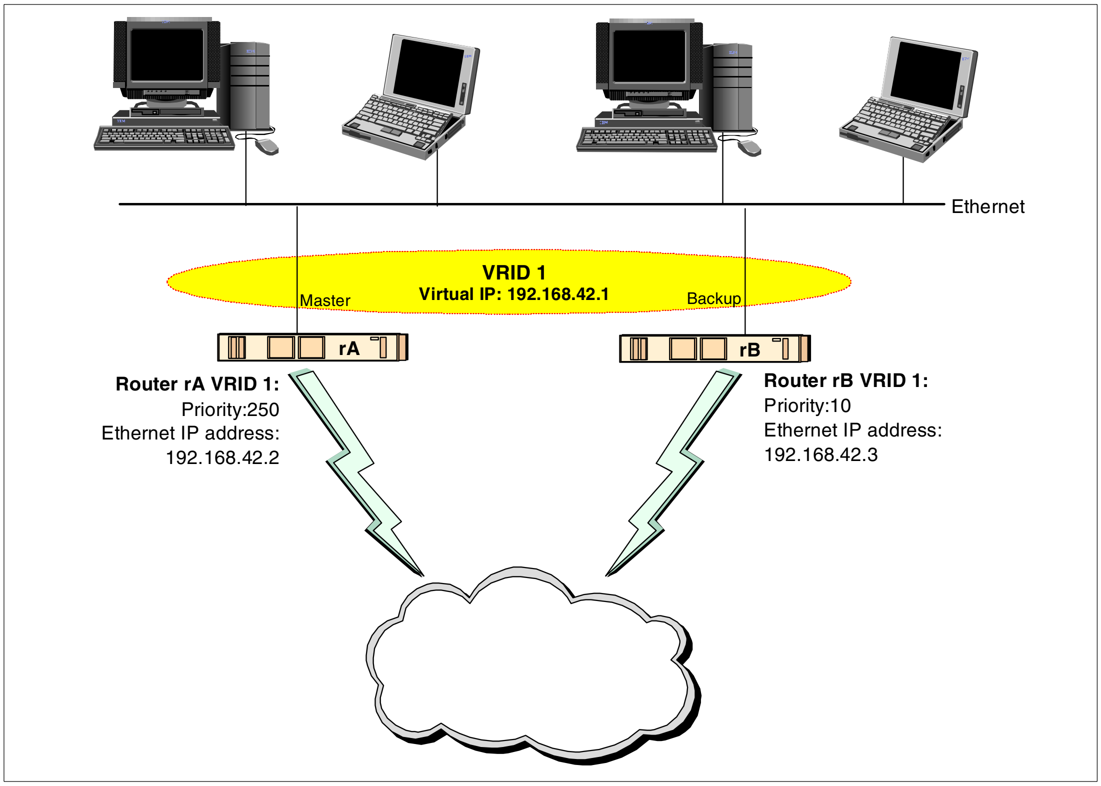


Figure 2 Simple VRRP example

When a VRRP instance is in the master state for a VRID, it sends multicast packets to the registered VRRP multicast address advising other VRRP instances that it is the master for the VRID. This serves two purposes:

- If a VRRP instance with a higher priority for that VRID is started (for example the VIP owner), the new VRRP instance can force an election and take on the master role.
- VRRP instances in the backup state for the VRID listen for the master’s packets; if an interval elapses without a packet being received, the instances in the backup state take action to elect a new master (since in all likelihood the master has failed).

In Figure 2, Router rA is the current Master for VRID 1. This means that one of the following must be true about Router rA:

- It is a higher priority than Router rB.
- It is the VIP owner (which mandates a priority of 255.;
- If both routers have the same priority, its interface IP address is higher than that of Router rB.

For our example, we have configured Router rA’s priority to be 250 and Router rB’s priority to be 10. If we did not set the priority on the routers (the default priority is 100), router rB would have become the master because the IP address of its Ethernet interface higher than that of router rA.

Note: The priority value actually has an additional use: it assists in some internal timing in the VRRP protocol. It is a good idea for your priority values to be at extremes, as it helps the protocol make “clean state” transitions. Refer to “Avoiding state transition race conditions” on page 31 for more detail on this.

Router failure

If Router rA suffered a failure, after a short interval Router rB would notice that no multicast packet had been received. It would then transition to the master state, taking over the handling of the VIP and sending its own multicast packets.

The length of time that Router rB waits before making its state transition is called the *master down interval*. It is based on the length of time between master updates (called the *advertisement interval*) and a value called *skew time* which is calculated from the priority value. Those curious about this kind of detail can find more information in “VRRP timing” on page 30.

Router restart

When the problem with Router rA is resolved, and depending on its configuration, either of two situations would occur:

- If Router rA is configured to start as Master, it will force an election immediately by sending its first advertisement as master. Router rB will receive this advertisement and transition to backup state.
- If Router rA is configured to start as Backup it will transition from initialization to Backup state. Nothing will happen until it receives an advertisement from Router rB. Router rB has a lower priority than Router rA, so Router rA will start an election by commencing its transition to master state and sending an advertisement. When it receives this advertisement, Router rB will transition to backup state because Router rA has higher priority.

Note: The VRRP RFC states that a VRRP instance should only transition to master state from startup if it is an owner. VRRP implementations can provide mechanisms to configure the initial state, but if Master is chosen when the VRRP instance is not an owner, then the configuration will not be operating in strict accordance with the RFC.

When planning your VRRP configuration, you know in advance which instance will be your preferred master as it will have the highest priority. Configuring the preferred master’s startup state allows it to transition straight to master when it is started, rather than waiting for advertisements from other instances.

If you do configure startup state, be careful to only configure one instance as master. This will avoid unnecessary elections when a lower priority instance starts up.

In both cases, Router rB signifies to Router rA that it has given up master state by sending an advertisement with a priority value of zero (0). When Router rA receives this, it knows that rB has deregistered the VRIP and that it can now successfully register the VRIP itself.

To find out more

See “VRRP details” on page 30 for more information about VRRP operation. For further details about the operation of VRRP (including frame formats and other information), refer to RFC2338 at:

```
http://www.faqs.org/rfcs/rfc2338.html
```

We also recommend the excellent book *VRRP: Increasing Reliability and Failover with the Virtual Router Redundancy Protocol*.

VRRP in a Linux on zSeries™ network

When building Linux guest environments under VM, we have the same routing issues as found in traditional server environments. It is not always desirable to have multiple network connections to each guest, and even if multiple connections were employed it becomes difficult to manage different paths to the network.

VRRP gives us a way to provide consistent and reliable routing service to Linux guests. By using VRRP in our Linux router guests, we present a single router image to Linux server guests.

> **Note:** It is perhaps unfortunate that Linux guests performing a routing function have been referred to as “virtual routers”, as this now creates a terminology overlap. We seem to be encountering a number of these in our Linux on zSeries work: we cannot refer to z/VM Guest LANs as “virtual LANs” because of IEEE 802.1q Virtual LAN technology; we discussed VIPA in a previous section; and now this one!

Linux VRRP implementations

While VRRP is implemented in many different router products, only three known implementations of VRRP exist for Linux.

Early VRRP implementations

The first VRRP implementations developed and available on Linux appear to have problems on VM Guest LANs.

Jerome Etienne’s VRRPd

This is perhaps the original implementation of VRRP for Linux. It is available at

```
http://off.net/~jme/vrrpd/index.html
```

The IBM® Redbook *Linux on IBM @server zSeries and S/390: Large Scale Linux Deployment*, SG24-6824, includes a brief discussion on this VRRP implementation.

On Guest LANs, this implementation does not function correctly because most of the actual network interface code has been hard-coded to work on Ethernet. It supports a mode that does not try to use the VRRP virtual MAC address, but even in this mode it fails to correctly transmit the VRRP multicast message packets. This means that the first instance that starts is able to operate as the VRRP router, but any others that start do not receive VRRP messages and try to transition to the master state.

Alexandre Cassen's VRRPd

Alexandre Cassen is a developer on the Linux Virtual Server project who took Jerome Etienne's VRRP code and started to develop it for use in LVS. His VRRP code is available from:

`http://linuxvirtualserver.org/~acassen/`

We did not actually try this VRRP implementation, as the author has extensive warnings about the "experimental", "proof-of-concept" nature of the code. Also, given that it is based on Jerome Etienne's implementation, we imagine that it will also be heavily dependent on Ethernet. Alexandre's follow-up program is much more interesting.

Keepalived

Alexandre Cassen merged his VRRP code into keepalived, a program he developed to provide heartbeat and other services to LVS. He added VRRP protocol support to keepalived because it provides a cooperative function to the other capabilities of keepalived.

The keepalived home page is located at:

`http://keepalived.sourceforge.net`

The VRRP implementation in keepalived appears to be sufficiently changed from its roots that it works without change on z/VM Guest LANs. This means that we can set up resilient routing within our Guest LAN environment without costly processing overhead in our Penguins.

Features of keepalived

To quote from the keepalived home page:

> The main goal of the keepalived project is to add a strong & robust keepalive facility to the Linux Virtual Server project... Keepalived implements a framework based on three family checks: Layer 3, Layer 4 & Layer 5/7... In addition keepalived implements an independent VRRPv2 stack to handle director failover.

In an LVS environment, keepalived can be used to provide health-checking of a server pool. It provides a way to keep the directors' LVS topology up-to-date in the event of a server failure. Those interested in the LVS application of keepalived should refer to the project home page for more information.

As stated earlier, the implementation of VRRP in keepalived is sufficiently hardware-neutral that it worked without modification on VM Guest LANs. We tested it on both QDIO and HiperSockets Guest LANs, under z/VM 4.3.

Note: For those who may be wondering, the VM APAR that provides MAC address capability in support of DHCP (described in *Linux on IBM @server zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP3596) was not applied to this z/VM system to make keepalived work. VRRP as implemented by keepalived does not depend on MAC addresses. Refer to "Virtual MAC address" on page 24 for more discussion on this.

Building and installing keepalived

We now describe the process of building and installing keepalived for Linux on zSeries and S/390®.

Dependencies

It is important to note that the VRRP protocol depends on multicast, so it will only work over network media that support multicast. In our case, this includes both QDIO and HiperSockets Guest LANs at z/VM Version 4 Release 3.

The README file for keepalived states that it is dependent on OpenSSL and popt.

OpenSSL

OpenSSL should be already available on your system if you have programs such as `sshd` installed. When building on SuSE SLES 7, however, compilation fails because of libraries that are missing from the distributed OpenSSL code.

To resolve this, edit the makefiles in the genhash, keepalived/core, and keepalived/vrrp directories to add defines for NO_RC5 and NO_IDEA. These defines will prevent the make process from trying to include the OpenSSL code that is not provided.

Note: Thanks to Adam Thornton (via the Linux-390 mailing list) for this workaround.

popt

The popt library provides routines to parse command line arguments. Check that popt has been installed before attempting to build keepalived.

Tip: Any RPM-based Linux distribution should have popt available, as RPM itself is dependent upon it.

ETHTOOL_GLINK

Part of the VRRP code in keepalived depends on an ETHTOOL IOCTL that is not implemented consistently in the Linux kernels on zSeries and S/390.

Important: The “ethtool” code in the Linux kernel must not be confused with the ethtool user-space program. Kernel-space ethtool provides IOCTL support for certain adapter functions (such as controlling full- or half-duplex communication) that can be controlled by applications such as user-space ethtool.

On a system using a stock 2.4.19 kernel (with s390 patches, of course), the keepalived code compiles cleanly and works well but experiences a startup delay problem. On systems from Red Hat (Red Hat Linux 7.2) and SuSE (SLES 7.0), using the vendors’ supplied kernels, compilation errors occur because the definition of the required IOCTL, ETHTOOL_GLINK, is missing from kernel-space ethtool.

Keepalived, as supplied, tests to see if the system is using a 2.4 kernel before including the code that uses ETHTOOL_GLINK. This assumes that all 2.4 kernels provide this IOCTL, which we have found not to be the case.

The patch shown in Example 1 on page 9 includes the ETHTOOL_GLINK code only if ETHTOOL_GLINK is actually defined. This allows keepalived to build correctly on stock kernels.

Example 1   Patch to keepalived for Linux/390 compatibility

```
diff -auNr keepalived-1.0.2.orig/keepalived/vrrp/vrrp_if.c keepalived-1.0.2/keepalived/vrrp/vrrp_if.c
--- keepalived-1.0.2.orig/keepalived/vrrp/vrrp_if.c     Mon Apr 14 10:29:00 2003
+++ keepalived-1.0.2/keepalived/vrrp/vrrp_if.c  Tue May 20 10:38:39 2003
@@ -183,7 +183,7 @@
 static int
 if_ethtool_status(const int fd)
 {
-#ifdef _KRNL_2_4_
+#ifdef ETHTOOL_GLINK
        struct ethtool_value edata;
        int err = 0;
```

Note: Thanks to Sebastian Korte (via the Linux-390 list) for alerting us to this issue.

Testing

While we have done basic testing to ensure that keepalived works as expected after these modifications are applied, we have not tested it exhaustively. Make sure you test keepalived to your satisfaction prior to relying on it in a production environment.

Updates

As always, staying in touch with the community on the LINUX-390 mailing list is a good way to keep up to date with developments and changes.

We will also be providing the information discussed here to the keepalived developers. Hopefully this will result in the modifications we require for Linux/390 and Guest LANs being incorporated into the keepalived source.

Building keepalived

Keepalived is available from the project home page. It can be downloaded in either SRPM (source RPM) format or tarball. A link is also provided to the Debian packages repository, although this seems to link only to packages for x86 systems.

Important: At the time of writing, the current keepalived version is 1.0.3. We did most of our testing with version 1.0.1, and filenames in the following examples reflect this.

Review the SuSE SLES 7 OpenSSL issue and the Linux/390 compatibility patch discussed in “Dependencies” on page 8 before building keepalived. If you are using a kernel from your Linux distributor (the kernels packaged with Red Hat 7.2 or SuSE SLES 7), you will need to make the changes discussed there prior to building the keepalived code.

Source tarball

To build keepalived from the source tarball, download the file from the project home page. Extract the tarball into an appropriate location using the following command:

```
# tar -zxf keepalived-1.0.1.tar.gz
```

This creates the directory keepalived-1.0.1 containing all the product files. Change into this directory, then issue the following commands:

```
# ./configure
# make
# make install
```

This will build and install keepalived on your system.

Important: We found that there was a minor bug in the installation script. The sample keepalived configuration file is supposed to be written to /etc/keepalived/keepalived.conf, but the script applies the base directory prefix to this—resulting in the file going to /usr/local/etc/keepalived/keepalived.conf.

To avoid this, you may wish to either alter the Makefile for the program, or manually relocate the /usr/local/etc/keepalived directory after installation.

Source RPM

Depending on your software maintenance practices, it may be preferable to build an RPM package from the source RPM and install that. To build keepalived from the source RPM, download the source RPM from the keepalived home page and install it with the following command:

```
# rpm -ivh keepalived-1.0.1-1.src.rpm
```

This puts the files that make up the keepalived package (in this case, just the source tarball) into the /usr/src/redhat/SOURCES directory. The SPEC file for the package, keepalived.spec, will be found in /usr/src/redhat/SPECS. This file specifies: how the package will be created; defaults for where the resulting product will be installed; and where configuration files will be written. Usually this file will not need to be edited, but it is always good to check.

Note: The main thing you might want to check or change is the location of the files installed. The keepalived.spec file that came with our source package uses the %{_bindir} and %{_sbindir} macros for the resulting binary files, meaning that the executables will go into /bin and /sbin directories.

You may also need to include modifications discussed in “Dependencies” on page 8. By including the patch in the SPEC file (you would need to write a suitable patch to the makefiles to resolve the SuSE SLES 7 OpenSSL issue), RPM will automatically apply the patch when it builds the binary package. RPM documentation can help you make the required changes.

Once you are happy with the SPEC file, build the keepalived RPM by issuing the following command:

```
# rpmbuild -bb /usr/src/redhat/SPECS/keepalived.spec
```

Once the package is built, you can install it with this command:

```
# rpm -ivh /usr/src/redhat/RPMS/s390/keepalived-1.0.1-1.s390.rpm
```

Running keepalived

Once keepalived is installed on your system, you will have to configure it. You will also need to make sure that keepalived is added to your system startup.

Command line

The keepalived program takes a number of options on the command line. The ones relevant to general operation or VRRP function are listed in Table 1 on page 11.

Table 1 Keepalived command line options

Option	Function
-V	Leave VRRP VRIPs registered when keepalived terminates
-n	Run keepalived in the foreground (don’t fork)
-f *filename*	Path to the configuration file (instead of default)
-h	Display usage information
-l	Log messages to local console (syslog)
-v	Display version information

Configuration

Keepalived uses a single configuration file, keepalived.conf, by default located in /etc/keepalived. A sample configuration, which could be used for Router rA in Figure 2 on page 4, appears in Example 2.

Example 2 keepalived.conf

```
vrrp_instance GLAN-1 {
    state MASTER
    interface eth0
    virtual_router_id 1
    priority 250
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1234
    }
    virtual_ipaddress {
        192.168.42.1/24
    }
}
```

Following are the key parameters in the configuration:

`vrrp_instance GLAN-1` Defines a new instance of VRRP.

`state MASTER` This VRRP instance will attempt to start in master state.

`interface eth0` This VRRP instance operates on the eth0 interface.

`virtual_router_id 1` The VRID this VRRP instance belongs to.

`priority 250` The priority of this VRRP router in this virtual router.

`advert_int 1` VRRP advertisements will occur every second (the default).

`authentication` Defines the type of authentication that will be used for advertisements on this VRID. `auth_type` can be either `PASS` for password authentication, or `AH` for IP Authentication Header method. `auth_pass` specifies the password that will be used.

`virtual_ipaddress` The list of VIPs that are managed by this VRID.

Startup

Whether you have installed keepalived from source tarball or RPM, you will need to make sure that it is started during system initialization. All the currently available Linux on zSeries distributions use SysV-style init scripts, so we recommend using this method. Use the following example to create an init script for keepalived.

Example 3 keepalived init script

```
#!/bin/bash
#
# chkconfig: 2345 19 81
# description: A status monitor and VRRP daemon
#
# processname: keepalived
# config: /etc/keepalived/keepalived.conf

# source function library
. /etc/rc.d/init.d/functions

# Get network config
. /etc/sysconfig/network

# Check that networking is up.
[ "${NETWORKING}" = "no" ] && exit 0

# The process must be configured first.
[ -f /etc/keepalived/keepalived.conf ] || exit 0

RETVAL=0

prog="keepalived"

case "$1" in
  start)
        echo -n $"Starting $prog: "
        daemon /usr/sbin/keepalived -l -f /etc/keepalived/keepalived.conf
        RETVAL=$?
        [ $RETVAL -eq 0 ] && touch /var/lock/subsys/keepalived
        echo
        ;;
  stop)
        echo -n $"Shutting down $prog: "
        killproc keepalived
        RETVAL=$?
        [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/keepalived
        echo
        ;;
  restart|reload)
        $0 stop
        $0 start
        RETVAL=$?
        ;;
  condrestart)
        if [ -f /var/lock/subsys/keepalived ]; then
                $0 stop
                $0 start
        fi
        RETVAL=$?
        ;;
  status)
        status keepalived
        RETVAL=$?
        ;;
  *)
```

```
        echo $"Usage: $0 {start|stop|restart|reload|condrestart|status}"
        exit 1
esac

exit $RETVAL
```

Important: This example script is based on a script supplied with a Red Hat distribution. SuSE has slightly different conventions for some parts of their scripts, so SuSE users should treat this as a guide only.

Also, a sample script is provided in the keepalived source package. It is contained in the etc/init.d directory of the keepalived source tree, in the file keepalived.init.

Once you have written the init script for keepalived, perform the necessary operations that will ensure the script is invoked at startup and shutdown. For example, on a Red Hat system the following command would accomplish this:

```
# chkconfig keepalived on
```

Alternatively, a utility such as ntsysv can be used to ensure that keepalived is started as required.

VRRP on Guest LANs using keepalived

In this section, we describe a number of ways that VRRP can be used to support IP routing in a Penguin Colony. We will start by converting the example in Figure 2 on page 4 to one that illustrates networking in a z/VM Linux environment, then employ VRRP and other techniques to improve the quality of IP routing service.

Simple environment without VRRP

First, let's examine a basic network configuration with hardware redundancy, as shown in Figure 3.

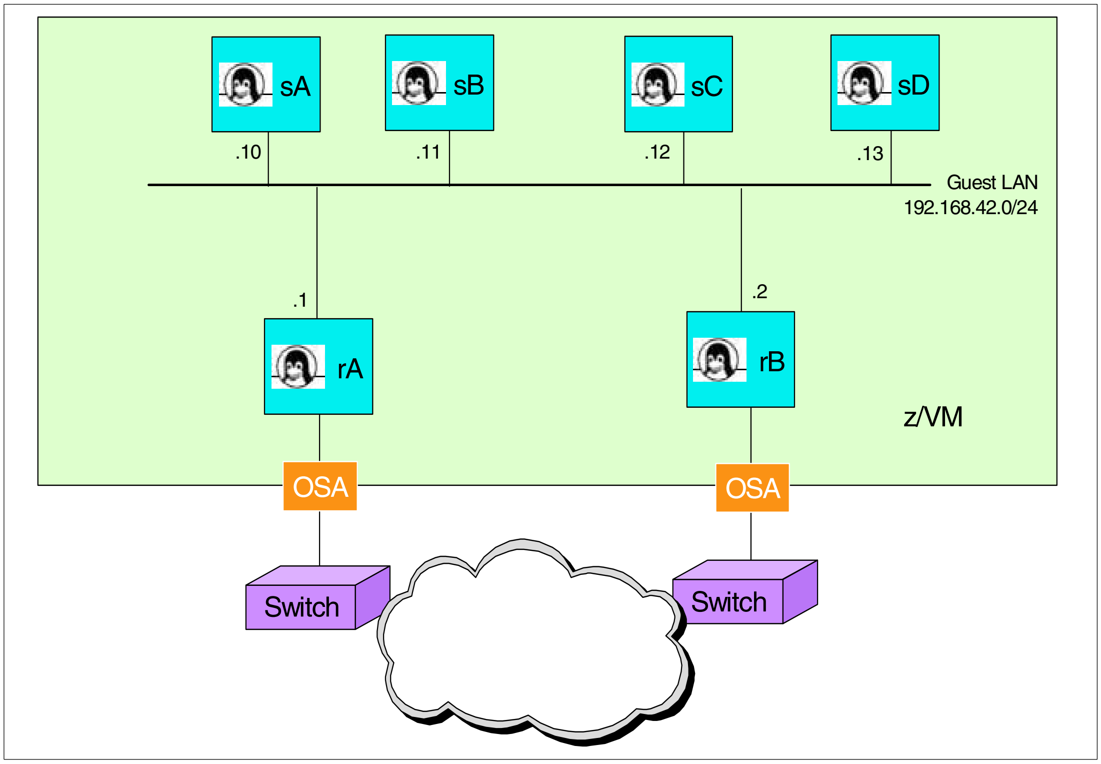


Figure 3 Linux routing configuration before VRRP

In this configuration, we have used some kind of dynamic routing protocol (OSPF, RIP, and so on) to provide routing redundancy into our Penguin Colony. We have not provided any way for the Linux guests to take advantage of the dual network paths, however.

In our guests we define a default gateway, which will be either one of the two router guests. If that router guest fails, traffic will be able to reach the Penguin Colony thanks to the dynamic routing in the network, but responses from our Linux guests will not reach the network.

Basic VRRP configuration

If we add VRRP to the example in Figure 3, we can provide redundancy for outgoing traffic. This is shown in Figure 4 on page 15.

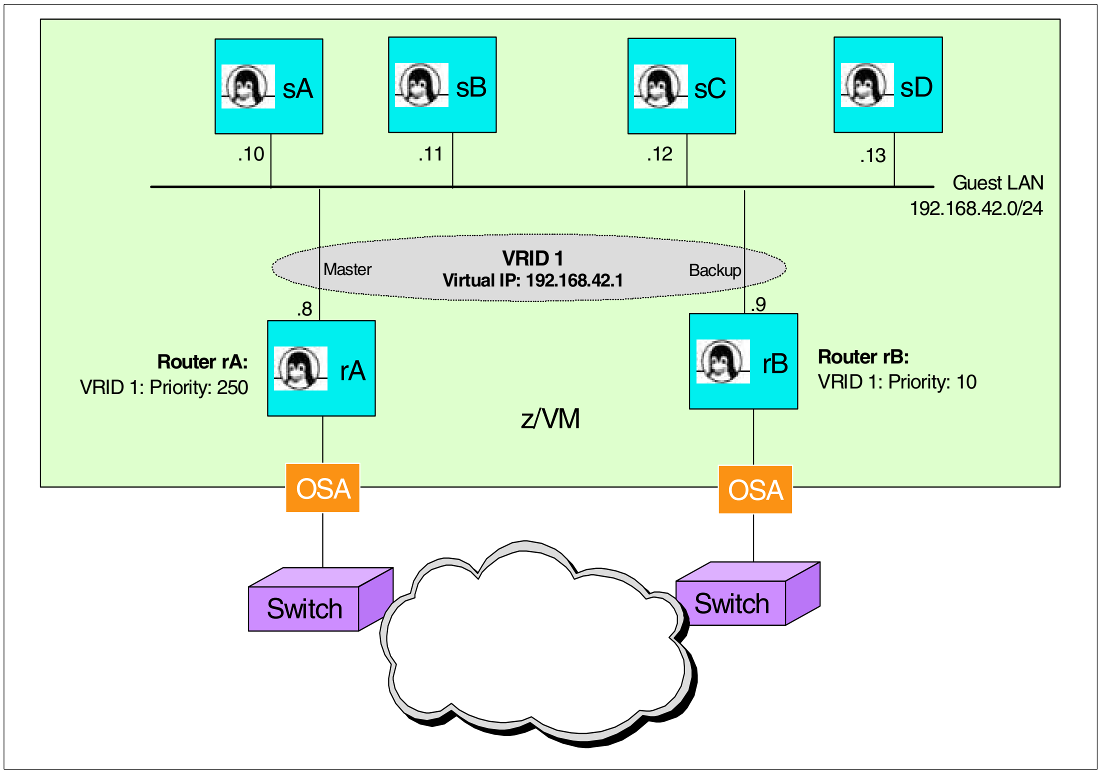


Figure 4 Linux routing with basic VRRP

In the configuration of the Linux guests, we specify the VIP as the default gateway. Now, VRRP will provide a continuous router service across the two routers.

> **Attention:** It is critical that the VIP is specified as the default gateway for the Linux guests, as it should be clear that VRRP does not protect the interface addresses of the routers. Any Linux guest that uses a router interface IP address will lose connectivity if that router fails, regardless of whether VRRP is working.
>
> As stated in “VRRP terminology” on page 3, an “owner” VRRP instance defines its VIP to be the same as a router interface IP address. While it is possible to specify the default gateway as an owner VRRP instance, we do not recommend it. Complications with VRRP owners are discussed in “Ownership” on page 32.

Configuration

The keepalived.conf that provides this function on Router rA is shown in Example 4 on page 16.

Example 4 Basic keepalived.conf - Router rA

```
! Configuration File for keepalived

vrrp_instance G-LAN {
    state MASTER
    interface eth1
    virtual_router_id 1
    priority 250
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1234
    }
    virtual_ipaddress {
        192.168.42.1/24
    }
}
```

Router rB's keepalived configuration is shown in Example 5.

Example 5 Basic keepalived.conf - Router rB

```
! Configuration File for keepalived

vrrp_instance G-LAN {
    state BACKUP
    interface eth1
    virtual_router_id 1
    priority 10
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1234
    }
    virtual_ipaddress {
        192.168.42.1/24
    }
}
```

Operation

When keepalived on router rA starts, it will attempt to transition to the master state because it has been configured to do so. It will set the VRIP address as a secondary address on the correct network interface, then start sending multicast advertisements to the registered VRRP multicast address (224.0.0.18).

When router rB starts, it will transition to backup state according to its configuration. It will set the master down interval according to the formula specified in the RFC, and wait to receive advertisements from the master.

Under z/VM, we can issue some CP commands that show us the IP addresses registered in a given Guest LAN. Example 6 on page 17 shows the output of the CP QUERY LAN DETAIL command.

Example 6 CP QUERY LAN DETAIL

```
QUERY LAN DETAIL G-LAN
LAN SYSTEM G-LAN       Type: QDIO     Active: 6     MAXCONN: INFINITE
  PERSISTENT  UNRESTRICTED  MFS: 8192    ACCOUNTING: OFF
    Adapter Owner: LINUXRA  NIC: 2210  Name: G-LAN
      192.168.42.8    192.168.42.1    224.0.0.1
      224.0.0.5       224.0.0.6       224.0.0.18
    Adapter Owner: LINUXRB  NIC: 2210  Name: G-LAN
      192.168.42.9    224.0.0.1       224.0.0.5
      224.0.0.6       224.0.0.18
    Adapter Owner: LINUXSA  NIC: 1000  Name: G-LAN
      192.168.42.10   224.0.0.1
    Adapter Owner: LINUXSB  NIC: 1000  Name: G-LAN
      192.168.42.11   224.0.0.1
    Adapter Owner: LINUXSC  NIC: 1000  Name: G-LAN
      192.168.42.12   224.0.0.1
    Adapter Owner: LINUXSD  NIC: 1000  Name: G-LAN
      192.168.42.13   224.0.0.1
Ready; T=0.01/0.01 11:54:40
```

You can see that in this Guest LAN, the guest LINUXRA has registered a number of IP addresses to the Guest LAN, including the VRRP multicast address and the VRIP.

LINUXRB has also registered the VRRP multicast address; this allows it to receive the advertisements from the master.

Note: Even though it is generating advertisements rather than receiving them, the master still needs to listen on the VRRP multicast address. Why? Because another VRRP instance with a higher priority may start on another router, and the current master must be able to receive the advertisements from such a router.

Other multicast addresses appear in this list, but they are not related to VRRP:

Table 2 Multicast addresses

Address	DNS name	Purpose
224.0.0.1	ALL-SYSTEMS.MCAST.NET	All multicast-capable systems
224.0.0.5	OSPF-ALL.MCAST.NET	Systems running OSPF
224.0.0.6	OSPF-DSIG.MCAST.NET	Systems providing the OSPF Designated Router (DR) function

Load sharing with VRRP

In a given VRID, only one router can be master at one time. This means that no matter how many routers are configured into a VRRP group, only one is handling IP traffic at a time. In the network in Figure 4 on page 15, this means that only one of the routers, and one of the OSAs, will carry network traffic at a time.

By configuring another VRID on the same Guest LAN, and then dividing the Linux guests between the two, we can load balance the outgoing traffic as shown in Figure 5 on page 18.

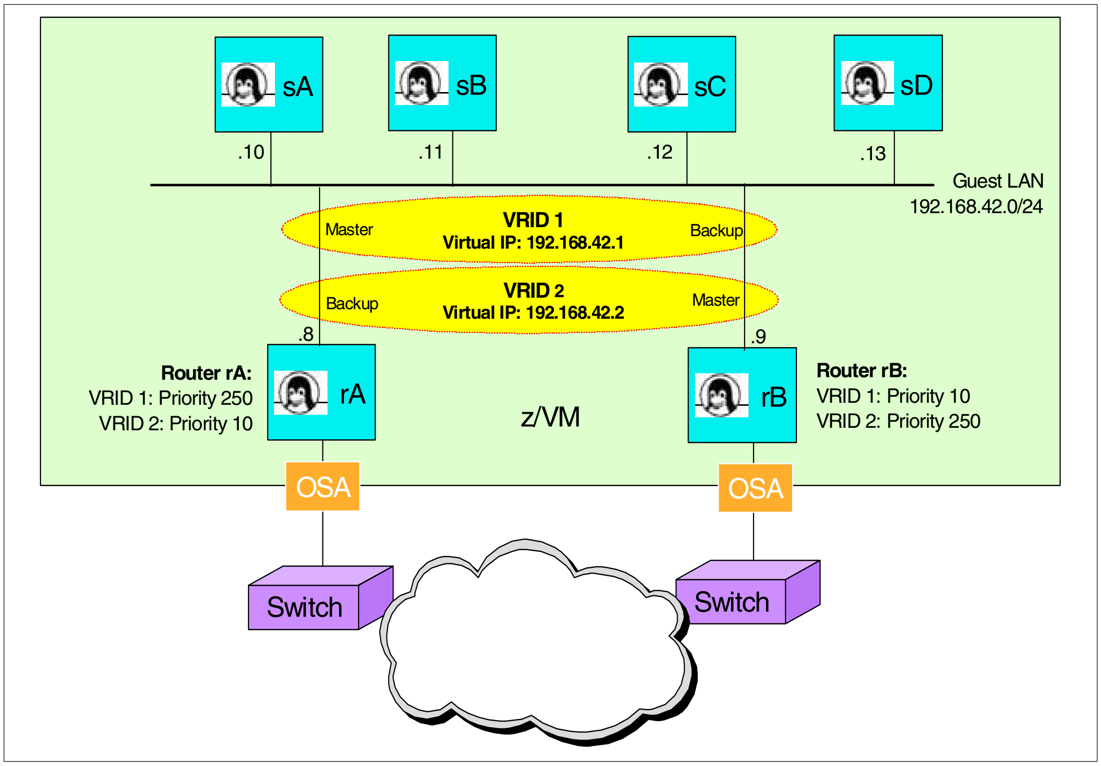


Figure 5 VRRP load balancing

If Router rA fails, there is no change to the VRID for which Router rB is the master. Router rB will take on master state for the other VRID, becoming master for both VRIDs. All guests continue to receive continuous routing service.

Configuration

The keepalived.conf that provides this function is shown in Example 7.

Example 7 Load sharing keepalived.conf - Router rA

```
! Configuration File for keepalived

vrrp_instance G-LAN-1 {
    state MASTER
    interface eth1
    virtual_router_id 1
    priority 250
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1234
    }
    virtual_ipaddress {
        192.168.42.1/24
    }
}

vrrp_instance G-LAN-2 {
    state BACKUP
    interface eth1
    virtual_router_id 2
    priority 10
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 4321
    }
    virtual_ipaddress {
        192.168.42.2/24
    }
}
```

Router rB’s keepalived configuration is shown in Example 8.

Example 8 Load sharing keepalived.conf - Router rB

```
! Configuration File for keepalived

vrrp_instance G-LAN-1 {
    state BACKUP
    interface eth1
    virtual_router_id 1
    priority 10
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1234
    }
    virtual_ipaddress {
        192.168.42.1/24
    }
}

vrrp_instance G-LAN-2 {
    state MASTER
    interface eth1
    virtual_router_id 2
    priority 250
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 4321
    }
    virtual_ipaddress {
        192.168.42.2/24
    }
}
```

Unfortunately, to take advantage of this configuration it is necessary to have a different default gateway specified on some of your Linux guests. This could easily be done, for example by using one VRIP for guests with an even IP address and the other VRIP for guests with odd IP address. However, this defeats the objective of having our guests configured as similarly as possible for cloning and other benefits.

It is probably more effective to employ this type of configuration when you have more than one Guest LAN handled by your router guests, as shown in Figure 6 on page 20. Here, all of the guests for a given Guest LAN are served by the one router, but the load of different LANs is distributed across the two routers.

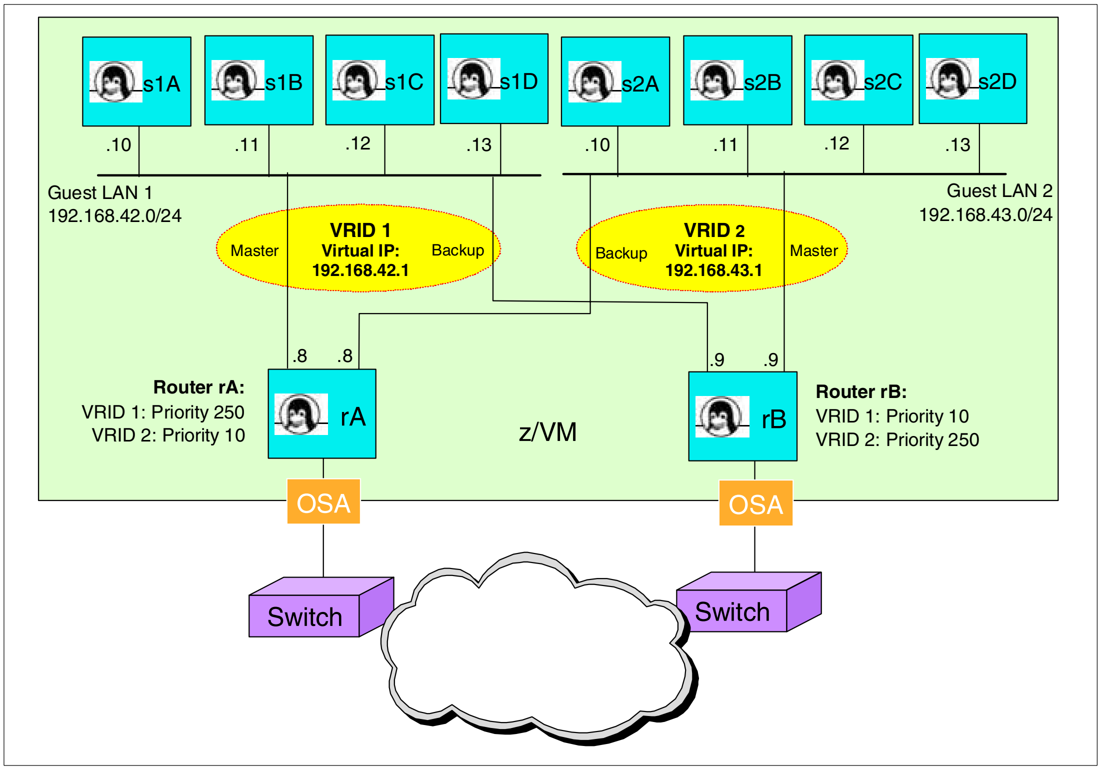


Figure 6 Load balancing multiple Guest LANs

This can be done with the configuration files changed as follows, first for Router rA (shown in Example 9):

Example 9 Improved load sharing keepalived.conf - Router rA

```
! Configuration File for keepalived

vrrp_instance G-LAN-1 {
    state MASTER
    interface eth1
    virtual_router_id 1
    priority 250
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1234
    }
    virtual_ipaddress {
        192.168.42.1/24
    }
}

vrrp_instance G-LAN-2 {
    state BACKUP
    interface eth2
    virtual_router_id 2
    priority 10
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 4321
    }
    virtual_ipaddress {
        192.168.43.1/24
    }
}
```

The configuration for router rB in this scenario is shown in Example 10.

Example 10 Improved load sharing keepalived.conf - Router rB

```
! Configuration File for keepalived

vrrp_instance G-LAN-1 {
    state BACKUP
    interface eth1
    virtual_router_id 1
    priority 10
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1234
    }
    virtual_ipaddress {
        192.168.42.1/24
    }
}

vrrp_instance G-LAN-2 {
    state MASTER
    interface eth2
    virtual_router_id 2
    priority 250
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 4321
    }
    virtual_ipaddress {
        192.168.43.1/24
    }
}
```

Using VRRP instead of dynamic routing

It is possible to use VRRP to provide routing capability for traffic coming into the Penguin Colony. This might be desirable if the use of a dynamic routing protocol is not available at your installation.

Because of the way it defines IP addresses, keepalived can be used on OSA Express adapters without having to use any of the “VIPA Takeover” functions provided by the QETH driver. This means that you can configure VRRP across your OSA Express interfaces and provide one or more VRIP addresses that your router network can use in static routes to reach your VM-internal simulated networks.

Figure 7 on page 22 shows an example of this. In this scenario, VRIDs protect both the Guest LAN and OSA interfaces of the two routers. The Linux guests use the internal VIP as a default gateway, and the network routers specify the external VRIP as the router to reach the Linux Guest LAN.

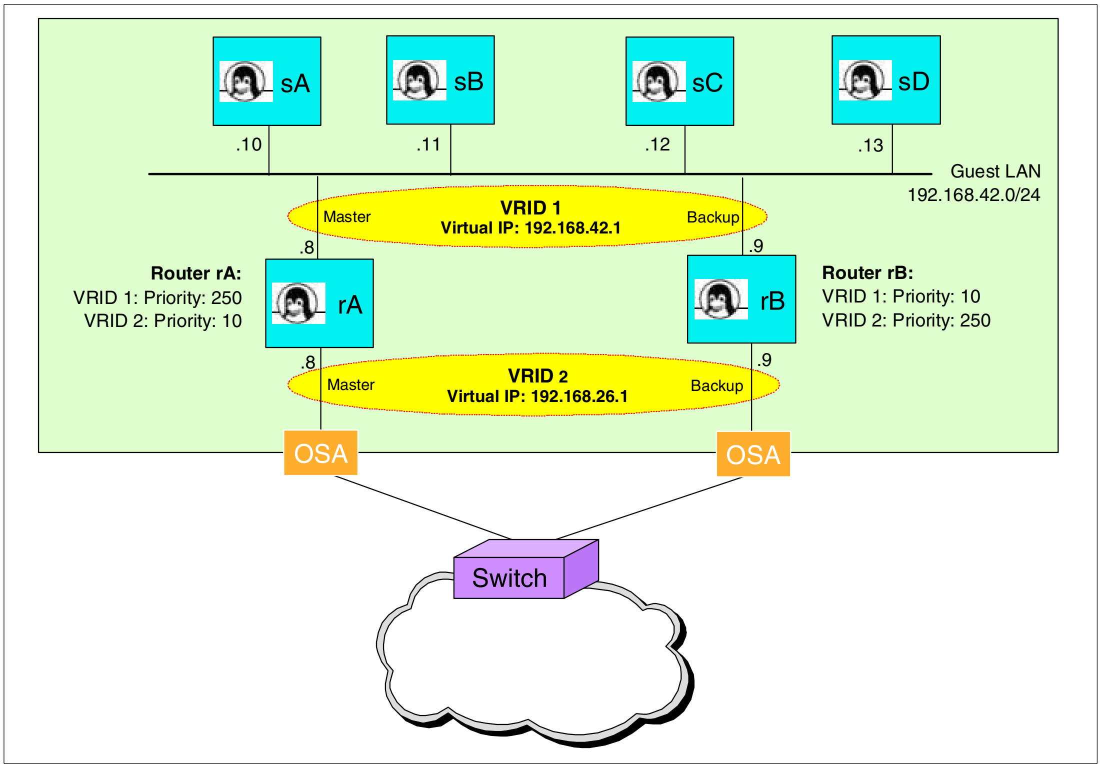


Figure 7 VRRP replacing dynamic routing

For many sites, this configuration may be sufficient to provide resilient routing both into and out of the Penguin Colony, without having to run a dynamic routing protocol in the Linux routers. It does, however, introduce some restrictions to your network configuration. It becomes necessary to have both external interfaces (OSA ports, in the example in Figure 7) connected to the same subnet[1]. Plus, if you have disaster recovery or other system configuration procedures that involve your system moving between processors or sites, the static route that points to your external VRIP address can cause problems.

Sync grouping

In addition, keepalived provides a feature known as “sync grouping”, that avoids the possibility of dead routing. In Figure 8 on page 23, we have used this feature to group the interfaces attached to the same router. If one of the interfaces in the group fails, all of the interfaces in the group are transitioned to failure mode. This avoids the problem where a router is master on one side, but the interface attaching it to a network on the other side is down.

[1] If your network uses IEEE 802.1q Virtual LANs, you can configure around the reduction in physical redundancy this would normally cause.

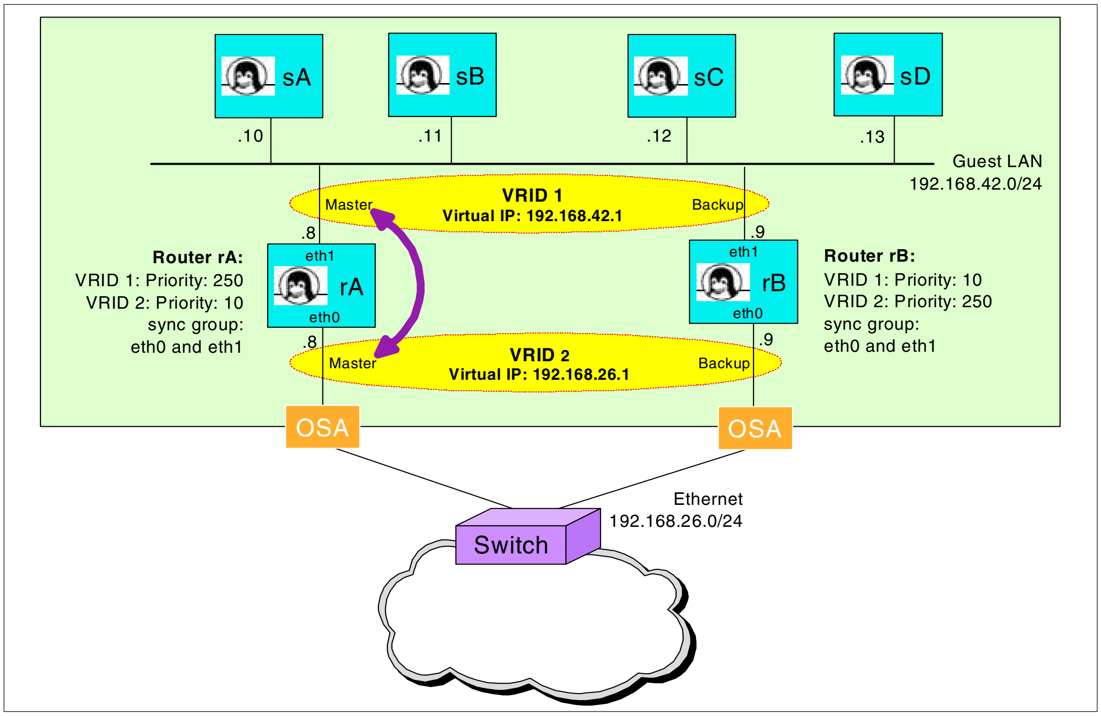


Figure 8 Grouping interfaces

When one interface in a sync group is in the master state, all other interfaces in that sync group will also be in master state. Likewise when one interface in the group fails, all the interfaces in the group are regarded as having failed. If the interface eth0 on Router rA failed, the result would appear as shown in Figure 9.

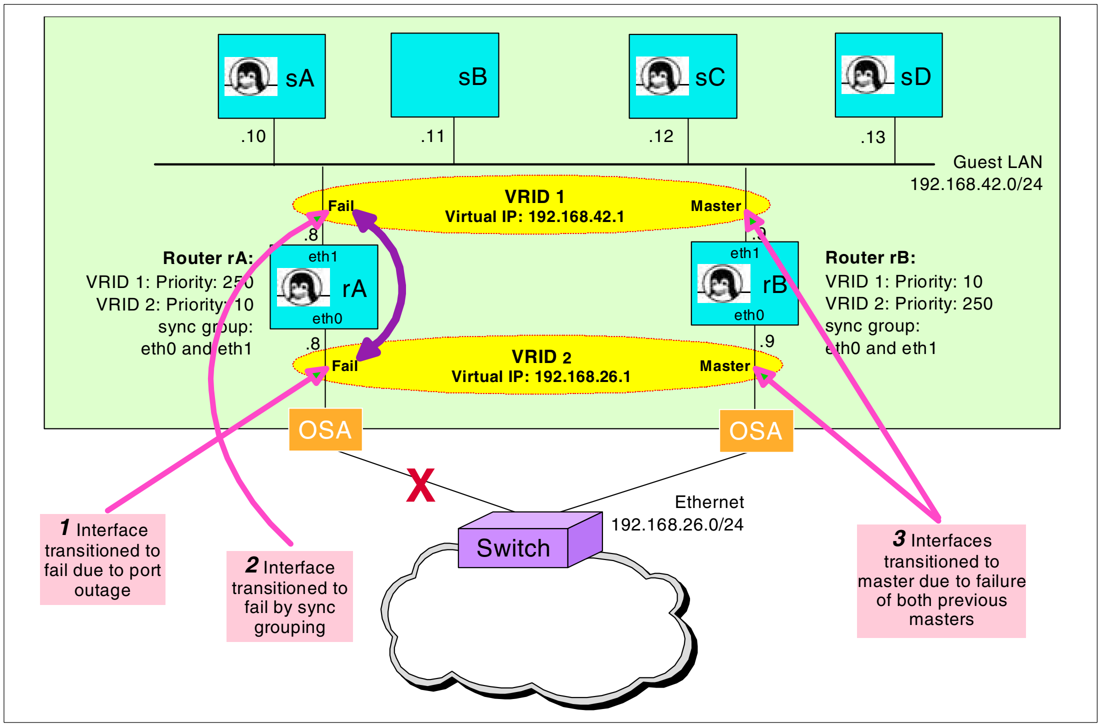


Figure 9 Interface grouping failure scenario

To understand why this feature is necessary, imagine the failure shown in Figure 9 on page 23 without sync grouping. In this case, Router rA would continue to be the master for VRID 1 even though it no longer has a path to the Ethernet network. This means that the Linux guests on the Guest LAN would no longer be able to reach the network. By forcing Router rA's VRID 1 instance to transition out of master state, router rB becomes the master for VRID 1 and we avoid this "routing into a dead network" situation.

Restriction: This feature exhibits an interesting failure mode, which we discuss in ""Unplugging" a simulated NIC" on page 28.

VRRP experiences on Guest LANs

In the following sections, we discuss some of the experiences we have had using VRRP on z/VM Guest LANs.

Virtual MAC address

The VRRP RFC states that proper VRRP operation on network media that use MAC addressing (such as Ethernet) requires that the VRRP instance must not use the router's network interface MAC address for transmitting VRRP packets, or for responding to ARP queries for the VRIP. Instead, a special MAC address must be used that is not related to the hardware address, but is a specially generated address from a range defined for VRRP.

The main reason this is done is to maintain the virtual nature of the virtual router. By using a hardware MAC address, there would be an association between the virtual router and the real hardware that is undesirable. If a state transition took place, clients may have cached the MAC address and be unable to reach the network, even though VRRP had done its work. By using a virtual MAC address, it does not matter if the clients cache the address because whichever router is the master will respond to the virtual MAC address.

In the original vrrpd, this virtual MAC address handling was done by changing the hardware MAC address of the Ethernet interface to the VRRP MAC address. Not only did this make the code Ethernet-dependent, but because the virtual MAC address is calculated using the VRID, only one virtual router could be handled by vrrpd at a time. To work around this, Jerome Etienne introduced an option to vrrpd to turn off the virtual MAC address processing. Now you could have more than one VRID per interface—but it was no longer RFC-compliant, and there was the possibility of additional downtime due to ARP caching in clients.

On z/VM Guest LANs, it is not possible to set the MAC address. Because of this, keepalived must be operated in the mode that turns off support for the virtual MAC address. On Guest LANs, though, this is not a reliability problem, because ARP processing (or its equivalent) is handled by CP. When a VRRP state transition occurs and a new router guest registers the VRIP, CP will ensure that from that moment on all traffic intended for that IP address will be directed to the right guest.

Important: VM APAR VM63172 changes the way that MAC addresses for Guest LAN interfaces are presented to Linux. It does not, however, change ARP processing in any way. ARP is still handled in CP by the Guest LAN code, so VRRP still functions in the same manner whether the PTF for this APAR is applied or not.

While you do not need the PTF for APAR VM63172 on your system to run keepalived, review the discussion in "Configuring an interface down" on page 28 for a situation where the fix resolved a VRRP issue we encountered during testing.

Interaction with Zebra

At this time, we recommend a combination of VRRP for internal routing and dynamic routing for the external network interface as the best way to provide highly redundant IP routing service for a z/VM Linux environment. Dynamic routing can be implemented on Linux using a number of different daemons, but one of the most popular is Zebra from the GNU Project.

We configured Zebra to use the OSPF routing protocol, and set up a test environment of Linux router guests to verify our configuration. A diagram of the configuration appears in Figure 10.

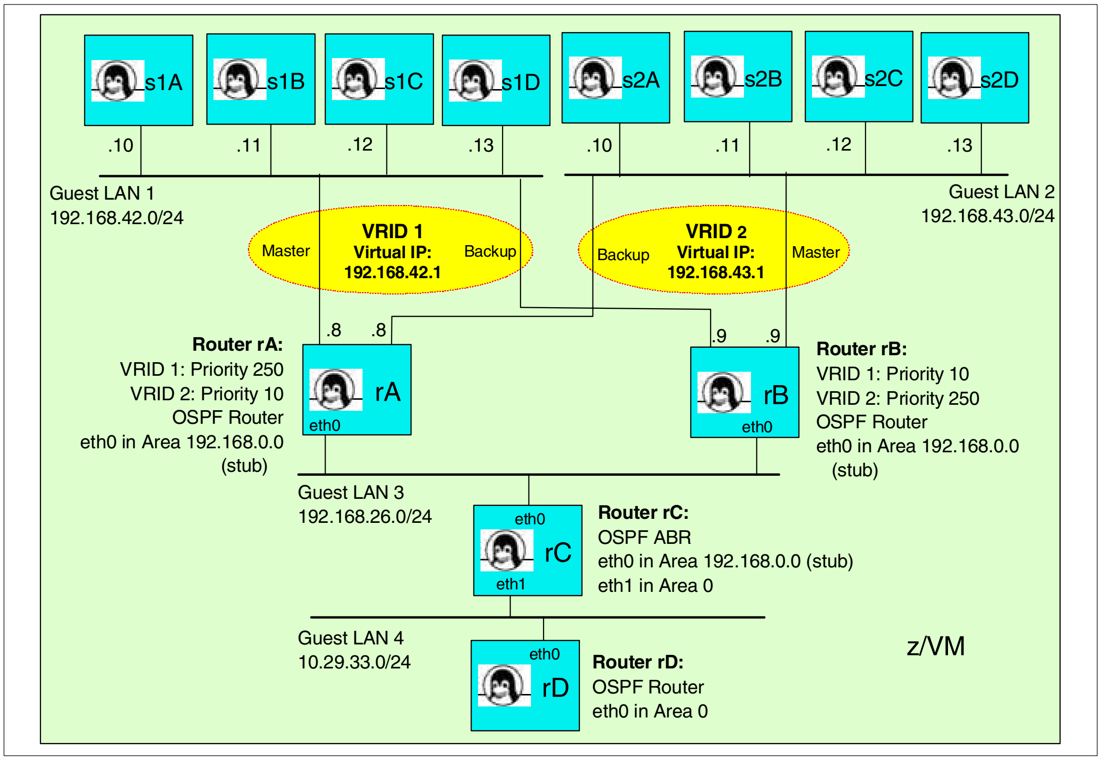


Figure 10 keepalived/Zebra interaction

This test was intended to show that Zebra correctly performed the Area Border Router (ABR) function (the boundary between an OSPF stub area and the backbone area) before we moved the testing into the real network using real hardware.

Tip: OSPF Stub Areas are ideal for the routers attaching your Penguin Colony to the rest of the network. Instead of learning about routes all over the network, the ABR injects a default route into the stub area, and can summarize the networks in the stub area into a single route for the backbone area. Routing tables on both sides of the boundary are simplified.

We verified that connectivity was as we expected, and sure enough we were able to reach all parts of the test setup as expected. We saw something unexpected in the route tables of the Zebra systems, however, which is shown in Example 11.

Example 11 IP details with “erroneous” host routes

```
[root@rC root]# ip route list
192.168.43.1 via 192.168.26.9 dev eth0  proto zebra  metric 2
192.168.42.1 via 192.168.26.8 dev eth0  proto zebra  metric 2
192.168.42.0/24 via 192.168.26.8 dev eth0  proto zebra  metric 2
192.168.43.0/24 via 192.168.26.8 dev eth0  proto zebra  metric 2
192.168.26.0/24 dev eth0  scope link
10.29.33.0/24 dev eth1  scope link
127.0.0.0/8 dev lo  scope link
```

The VRIP addresses were being imported into the OSPF routing domain as host routes, and being advertised to the other OSPF routers. The guests should have been able to contact the VRIP addresses directly via the local interfaces, but instead were being directed onto alternate interfaces and routing to the VRIPs.

Attention: We should add at this point that the interfaces facing the Guest LANs were defined to OSPF as passive interfaces. We do not want OSPF broadcasts going out over our Guest LANs. This is why the mystery routes appear against the routers’ “external” interfaces; these are the only OSPF interfaces the routers have.

Regardless, when you use OSPF in your Penguin Colony, always define the Guest LAN interfaces as passive interfaces; OSPF then knows that the networks exist, without sending OSPF messages into them.

After much confusion, and several unsuccessful attempts to filter the offending host routes from OSPF, we realized the address added by keepalived was actually an address with a host mask (rather than an additional subnet IP address using the network mask). This caused Zebra to import the address as a host route, instead of treating it as a secondary address.

Example 12 Address details from Router rA

```
[root@rA root]# ip address list
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
    inet6 ::1/128 scope host
2: sit0@NONE: <NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
3: eth2: <MULTICAST,UP> mtu 1492 qdisc pfifo_fast qlen 100
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.43.8/24 brd 192.168.43.255 scope global eth2
    inet6 fe80::200:ff:fe00:0/10 scope link
4: eth1: <MULTICAST,UP> mtu 1492 qdisc pfifo_fast qlen 100
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.42.8/24 brd 192.168.42.255 scope global eth1
    inet 192.168.42.1/32 scope global eth1
    inet6 fe80::200:ff:fe00:0/10 scope link
5: eth0: <MULTICAST,UP> mtu 1492 qdisc pfifo_fast qlen 100
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.26.1/24 brd 192.168.26.255 scope global eth0
    inet6 fe80::200:ff:fe00:0/10 scope link
```

We tried adding a subnet mask to the VRIP address definition in the keepalived.conf file, and were immediately successful. The resulting change in Router rA's address table is shown in Example 13.

Example 13 Corrected address details from rA

```
[root@rA root]# ip address list
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
    inet6 ::1/128 scope host
2: sit0@NONE: <NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
3: eth2: <MULTICAST,UP> mtu 1492 qdisc pfifo_fast qlen 100
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.43.8/24 brd 192.168.43.255 scope global eth2
    inet6 fe80::200:ff:fe00:0/10 scope link
4: eth1: <MULTICAST,UP> mtu 1492 qdisc pfifo_fast qlen 100
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.42.8/24 brd 192.168.42.255 scope global eth1
    inet 192.168.42.1/24 scope global secondary eth1
    inet6 fe80::200:ff:fe00:0/10 scope link
5: eth0: <MULTICAST,UP> mtu 1492 qdisc pfifo_fast qlen 100
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.26.1/24 brd 192.168.26.255 scope global eth0
    inet6 fe80::200:ff:fe00:0/10 scope link
```

Looking back to the route table on Router rC, we see that the host routes we saw before have disappeared.

Example 14 Expected route table of Router rC

```
[root@rC root]# ip route list
192.168.42.0/24 via 192.168.26.8 dev eth0  proto zebra  metric 2
192.168.43.0/24 via 192.168.26.8 dev eth0  proto zebra  metric 2
192.168.26.0/24 dev eth0  scope link
10.29.33.0/24 dev eth1  scope link
127.0.0.0/8 dev lo  scope link
```

We assumed that keepalived would pick up the subnet mask from the interface, and that we had found a bug or incompatibility that prevented keepalived from obtaining the correct subnet mask for Guest LAN interfaces. However, all the keepalived documentation we have seen shows VRIP addresses with "/32" masks. So it seems as though keepalived is "working as designed", behaving no differently on Guest LAN than it does on other media.

Note: All of the example keepalived configurations shown in this paper include the subnet mask specification. Having the addresses default to host masks does not seem to affect the operation of keepalived or the VRRP protocol, and neither does changing them to subnet masks. It is worth noting that the RFC is not clear on whether a host or subnet mask should be used for VRIP addresses.

We will leave it up to you to decide how you define your addresses, but recommend that if you are using a dynamic routing protocol, then define them with subnet masks, to avoid having unexpected route entries appear.

Failure modes

We wanted to verify that keepalived and the VRRP protocol would achieve the desired effect. We tested by running the ping program from one Linux guest to an address on the other side of the router, using the VRIP address as the gateway address.

The obvious failures (killing the keepalived daemon, FORCEing the router guest off VM, destroying the simulated NIC) all were handled as expected, with a backup VRRP instance taking over the master role and the traffic continuing on with only a dropped packet or two between the failure and the state transition.

However, a couple of failure modes we tested exhibited unexpected results, and we discuss them here.

Configuring an interface down

We tested using the **`ifdown`** command to bring down an interface with a VRRP instance running over it, to see the effect on the VRRP operation.

We found that the expected VRRP state transition occurs when the interface was brought down. When the interface was brought back up, however, the VRRP multicast address was not reregistered and VRRP failed.

Note: If a VRRP instance is operating without having registered the VRRP multicast address, it will not receive any advertisements sent by other VRRP instances.

This situation would be most disruptive when the instance that rejoins the virtual router is lower priority than the current master. When the new instance starts, the master down interval will pass without it receiving any messages from the master. This will cause it to attempt to transition to master, which will fail because the current master has already registered the VRIP. It will then start sending advertisements, which the current master will receive, process, and ignore (because they come from a lower priority instance).

It is unlikely that an outage would result from this, unless the current master then failed and there were no more backups (or the other backups were lower priority than the “failed” backup). Because the broken instance already thinks it is in master state, it will not attempt to reregister the VRIP. Even if there are other backups of lower priority, they will receive the advertisements from the broken instance and believe that it is handling the VRIP.

Exercise: keepalived also provides a heartbeat facility. It would be interesting to see if it could be configured to monitor the VRIP, thus becoming a self-monitoring program!

We noticed that the change list on the latest version of the Linux OCO modules included reference to a bug in multicast group membership registration. We verified that we were running the latest version of the OCO drivers, but still experienced the failure.

Later, our system was maintained and the fix for VM APAR VM63172 was applied. We performed this test again, and this time found that the multicast memberships were correctly reregistered after the interface was configured back up. So, it would appear that the fix included in the qeth module is dependent upon a corresponding fix in the z/VM Guest LAN code made available with the PTF for APAR VM63172.

“Unplugging” a simulated NIC

When a simulated NIC is detached from a Guest LAN but *not* destroyed, the Guest LAN code represents this to the QETH driver as a cable pull event[2]. The NIC can no longer pass traffic, so if this is done to a NIC carrying a VRRP instance in master state, the backups will no longer receive any advertisements and an election will occur.

Linux still believes the interface is up, so keepalived will attempt to transition the instance to master because no advertisements are being received. Because the “cable is unplugged”, however, it does not matter if that instance attempts to become master again, because any advertisements it sends will not reach the network.

The problem with this situation occurs when sync grouping is used. When the NIC is first detached, a backup wins the resulting election and transitions to master. Now, because of sync grouping, the other interface on this router transitions to master also. The advertisements it sends are received by the still-attached interface on the first router, which will start to transition back to master because the multicast it has received from the second router is lower priority. The second router returns to backup state, but again does not receive advertisements from the first router due to the detached NIC. The cycle continues until the NIC on the first router is reattached.

Note that the same problem will still occur if the NIC is detached on a backup VRRP instance; it will attempt to transition to master when it no longer receives advertisements over the detached NIC.

If detaching NICs is something you are likely to do in your environment, take care when using sync grouping. To prevent this situation, shut down keepalived on the router that uses the NIC you wish to detach. If the router is a master, a backup will transition cleanly to master. If the router is a backup, it will no longer be participating in the virtual routers and will not mistakenly attempt to transition to master.

AH troubleshooting

We tested the use of stronger authentication through the use of keepalived’s Authentication Header (AH) support.

Note: AH is an IP standard, defined in RFC2402, that adds additional header information to IP packets. This additional header provides protection against the alteration and replay of packets, which is particularly useful in VRRP because TCP connections are not used.

Starting the keepalived processes on our two test router guests resulted in normal operation of the VRRP virtual routers. As expected, the additional data payload resulted in slightly more traffic over the network, as shown in Figure 11 on page 31.

On a couple of occasions, however, after the restart of one (sometimes both) of the keepalived processes, keepalived would fail to synchronize the AH sequence numbers. This would result in a massive amount of network traffic being generated, as both the router guests started flooding advertisements with unsynchronized AH sequence numbers into the network.

It would appear that the AH implementation in keepalived needs some more work. For the time being, we would recommend using PASS authentication in keepalived, and using other means to protect the virtual routers from bogus advertisements.

Tip: The Keepalived Project has written a paper to the IETF on their AH implementation. This would be an ideal place to start to find out more about how keepalived implements AH, and how this synchronization issue might be addressed. The paper is available from the keepalived project home page.

[2] A fairly interesting, but quite logical, thing to do.

VRRP details

In this section, we outline some of the detail involved in VRRP implementations.

VRRP timing

There are two critical time intervals in VRRP: the *advertisement interval* and the *master down interval*.

Advertisement interval

The advertisement interval is the length of time between the advertisement packets sent by the master VRRP instance. By default this is one second, but it is configurable.

The value chosen for the advertisement interval is a compromise between the amount of network traffic generated (and the amount of CPU used by router images and Guest LANs) and the possible downtime incurred through router failure. See “Choosing your advertisement interval” on page 30 for more information on this.

Master down Interval

The master down interval is the length of time that a backup VRRP instance will wait for an advertisement before attempting to transition to the master state. The VRRP RFC defines the master down interval as equal to three times the advertisement interval plus the skew time, which means it cannot be set by itself but is instead derived from the advertisement interval and the priority value.

Skew time

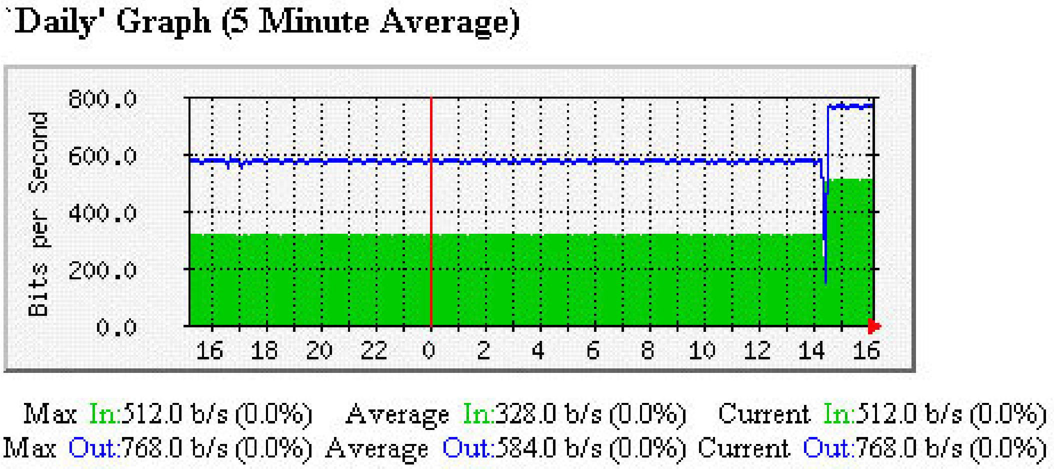
The VRRP RFC defines a value called the *skew time*. This value is calculated as follows:

$$skewtime = \frac{256 - priority}{256}$$

The result is a value, in seconds, which gets smaller as priority becomes larger. This value helps avoid race conditions between backup VRRP routers, as described in the following section.

Choosing your advertisement interval

When building your VRRP configuration, it is very important to ensure that a balance is reached between system overhead and backup capability. Figure 11 on page 31 shows an MRTG graph of the traffic placed on a HiperSockets network by two VRRP routers each supporting two VRRP instances (a configuration shown in Figure 5 on page 18).



Max In:512.0 b/s (0.0%) Average In:328.0 b/s (0.0%) Current In:512.0 b/s (0.0%)
Max Out:768.0 b/s (0.0%) Average Out:584.0 b/s (0.0%) Current Out:768.0 b/s (0.0%)

Figure 11 MRTG graph of traffic load

Note: The increase in the amount of traffic at approximately 2:30 pm shown on this graph resulted from a change in authentication method from PASS to AH. If you are considering using AH, you may wish to take this increase in traffic into account.

In this configuration, we used the default advertisement interval of one second. If the advertisement interval were increased to two seconds, not only would the amount of network traffic be halved, but the amount of CPU used by the Linux guests and the Guest LAN code would also be reduced.

Increasing the advertisement interval, however, will have a corresponding effect on the ability of your backup VRRP instances to respond to the failure of the master. Because the master down interval is calculated by multiplying the advertisement interval by three (and adding the skew time), doubling the advertisement interval will double the master down interval. So, instead of taking roughly three seconds to respond to failure of the master, with an advertisement interval of two seconds it would now take more than six seconds for a backup to take over master state.

If your installation can accept a routing infrastructure that is less reactive to outage, increasing the advertisement interval is the best way to reduce the overhead of VRRP.

Avoiding state transition race conditions

The addition of the skew time value into the master down interval helps the protocol avoid a situation where a number of backup VRRP routers attempt to transition to master state at the same time due to the failure of the previous master.

When a VRRP instance is in backup state, its operation is controlled by the arrival of advertisements from the master. The arrival of an advertisement resets the master down timer, a timer that the backup uses to determine if the master is still functioning. The master down timer pops at the end of the master down interval (see “Master down Interval” on page 30), and when this timer pops, the backup knows that there is a problem with the master and it commences a transition to master.

A race condition could arise as a result of all backups being “clocked” by the same advertisement from the master. When the master down timer pops, it would pop at the same time on all the VRRP instances that are part of that VRID. This means that all the backups would attempt to transition to master simultaneously, resulting in a flood of advertisements onto the network containing conflicting information. The condition would eventually settle down as the VRRP instances used priority values and IP interface addresses to decide the new master, but the confusion might have led to extended outage time, extra CPU usage, and extra network traffic generated.

Using skew time

The designers of VRRP had to ensure that the backup VRRP instances did not attempt to transition to master at the same time. In addition, they wanted to ensure that a higher priority instance was more likely to transition to master than a lower priority instance. The mechanism they arrived at introduced a value known as *skew time*, described in "Skew time" on page 30.

The skew time value is used as part of the calculation of the master down interval. The amount of skew time decreases as the priority increases. This means that a lower priority router will have a longer master down interval, and so will wait longer for an advertisement to arrive. The longer a low priority router waits, the more likely a higher priority router will transition to master state ahead of it (since its skew time will be shorter).

For this reason, it is important that priority values between VRRP instances are largely different. A difference of 240 between priority values (one router with priority 250 and another with priority 10) will yield a difference in skew times of 938 milliseconds. This means that the high-priority router will have made transition to the master state almost a full second before the low priority router would attempt to. However, if one router was priority 250 and the other was priority 249 (for example), the higher-priority router would transition to master state only 4 milliseconds before the lower priority router. This could well give rise to a race condition.

Note: A very thorough explanation of skew time deltas and other issues is given in *VRRP: Increasing Reliability and Failover with the Virtual Router Redundancy Protocol*.

Ownership

We mentioned in "Basic VRRP configuration" on page 14 that it is possible for the configured interface address of one of the routers running a VRRP instance to be the same as the VRIP. When this occurs, that VRRP instance is called the *owner* of the VRIP. When this is the case, VRRP dictates that as long as the VRRP instance is running, that instance must be master for that VRID and its priority must be set to 255.

There are issues with this type of configuration, however; it complicates management of the router, and potentially could cause unexpected downtime.

Note: Both restrictions mentioned here are peculiar to the current Linux implementations of VRRP. The different capabilities of router hardware (and the software that operates it) allow for more direct control of network data frames, making router-based VRRP implementations more "accurate" in terms of the RFC.

Connectivity to VRIP owner

If the VRIP is used as a destination address for connecting to the router, unpredictable results will occur if a state transition happens during the session. The connection will either fail (if a TCP application, such as SSH), or else unexpected errors or bad data will result (for UDP-based applications such as SNMP, or for TCP applications that use many short sessions, such as HTTP). If the owner is down, management applications will be accessing one of the backup routers without providing any awareness that they are not accessing the intended router.

Restart after failure

If a backup VRRP instance has taken up the master state after the failure of the owner, all is working normally. When the owner restarts, however, it will not be able to set its IP address because the backup has configured it. The owner's IP stack will not permit the configuration of a duplicate IP address. This could require manual intervention (and a brief outage) to resolve; the resolution could be difficult too, because the router will be inaccessible from the network since its IP address could not be configured.

References

ITSO publications

- *Linux on IBM @server zSeries and S/390: Large Scale Linux Deployment*, SG24-6824

  http://www.ibm.com/redbooks/abstracts/sg246824.html
- *Linux on IBM @server zSeries and S/390: TCP/IP broadcast on z/VM Guest LAN*, REDP3596

  http://www.ibm.com/redbooks/abstracts/redp3596.html

Other resources

- *VRRP: Increasing Reliability and Failover with the Virtual Router Redundancy Protocol*, Ayikudy Srikanth and Adnan Adam Onart, Addison-Wesley, 2003, ISBN 0-201-71500-7

Referenced Web sites

- Virtual Router Redundancy Protocol RFC

  http://www.faqs.org/rfcs/rfc2338.html
- The VVRPd home page

  http://off.net/~jme/vrrpd/index.html
- Alexandre Cassen's software patches

  http://linuxvirtualserver.org/~acassen/
- The keepalived home page

  http://keepalived.sourceforge.net

About the author

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

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

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

- Greg Geiselhart, ITSO Poughkeepsie
- Mike Brady of `devnull.net.nz` for technical assistance
- The administrators and members of the Linux-390 mailing list

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.

This document created or updated on June 10, 2003.

Send us your comments in one of the following ways:

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

Trademarks

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

@server™
Redbooks™
Redbooks(logo) ™
IBM®
S/390®
z/OS™
z/VM™
zSeries™

The following terms are trademarks of other companies:

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.

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

MMX, Pentium and ProShare are trademarks of Intel Corporation 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.