User Guide

Alexandre Cassen

http://www.keepalived.org

acassen@linux-vs.org

Licence

CONTENTS

I. Introduction

Load balancing is a good solution for service virtualization. When you design a load balanced topology one must take special care of:

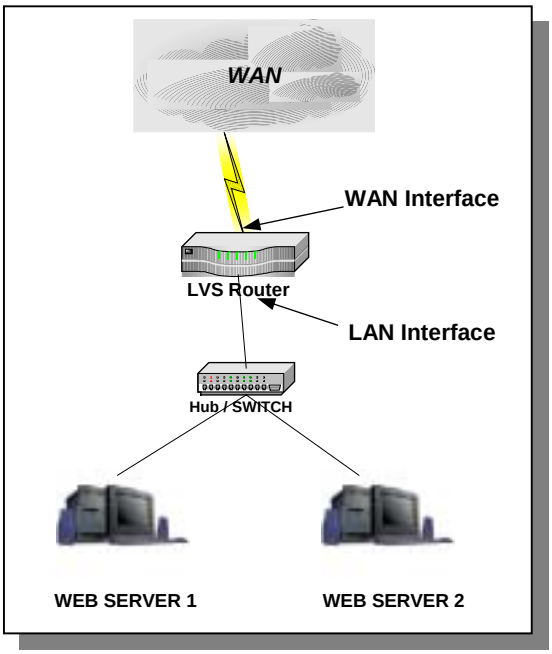
- Real server availability using health-checks.
- Load balancer availability using failover protocol.

Load balancing real services, provides a global Highly Available virtual service. To increase the load balanced service availability we need to monitor each real server node. This problem is mainly handled using a health-check framework manipulating a real server pool.

On the other hand, when using a load balancer director we introduce a Single Point Of Failure for the virtual service. So load balancer high availability must also be handled, using dedicated routing protocols for director failover/virtualization.

Keepalived tries to address these two problems by adding, on the one hand, a strong & robust health-check framework, and on the other hand, implementing a Hot Standby protocol. These two frameworks can deal with the Linux Virtual Server (LVS) framework to manipulate LVS real server pools by adding or removing real servers based on health-checks' decisions.

II. Terminology



LVS stands for "Linux Virtual Server". LVS is a patched Linux kernel that adds a load balancing facility. For more information on LVS, please refer to the project homepage: http://www.linux-vs.org. LVS acts as a network bridge (using NAT) to load balance TCP/UDP stream. The LVS router components are:

- *WAN Interface*: Ethernet Network Interface Controller that will be accessed by all the clients.
- *LAN Interface*: Ethernet Network Interface Controller to manage all the load balanced servers.
- *Linux kernel*: The kernel is patched with the latest LVS and is used as a router OS.

In this document, we will use the following keywords:

LVS component:

- ***VIP***: The Virtual IP is the IP address that will be accessed by all the clients. The clients only access this IP address.
- ***Real server***: A real server hosts the application accessed by client requests. WEB SERVER 1 & WEB SERVER 2 in our synopsis.
- ***Server pool***: A farm of real servers.

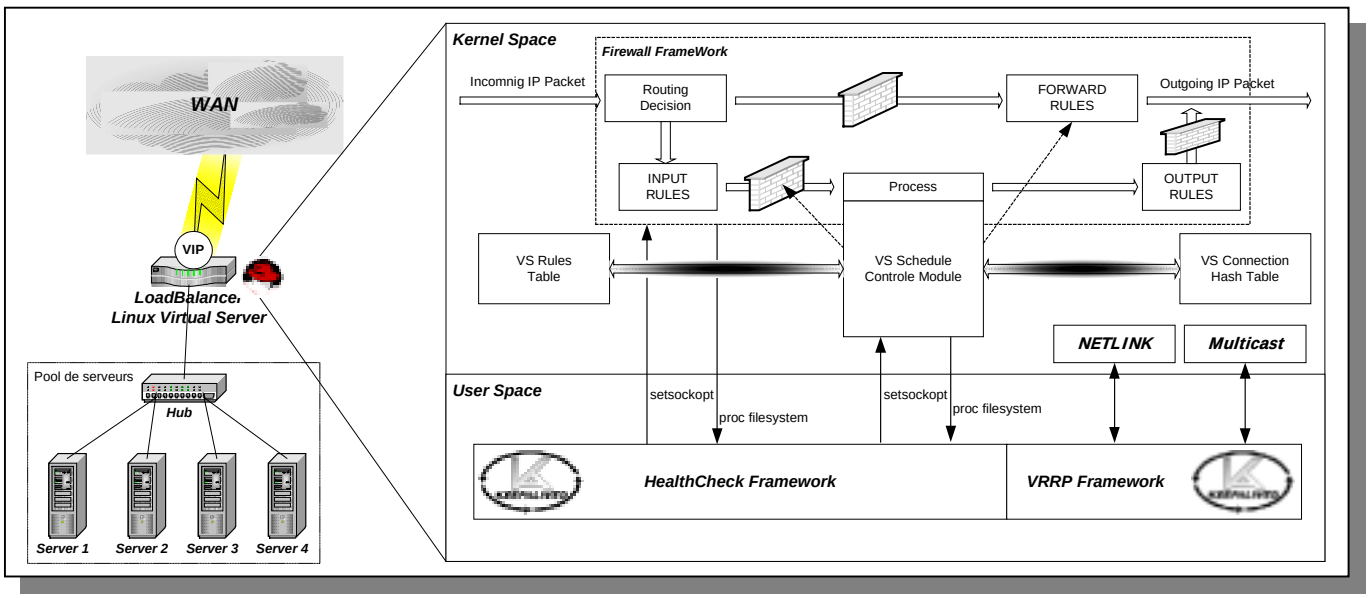
- ***Virtual server***: The access point to a Server pool.
- ***Virtual Service***: A TCP/UDP service associated with the VIP.

VRRP component:

- ***VRRP***: The protocol implemented for the directors' failover/virtualization.
- ***VRRP Instance***: A thread manipulating VRRPv2 specific set of IP addresses. A VRRP Instance may backup one or more VRRP Instance. In our "Case study: Failover", we are dealing with 4 VRRP Instances. One owning (VIP1,VIP2), one owning (VIP3,VIP4), one owning (DIP1) and one owning (DIP2). It may participate in one or more virtual routers.
- ***IP Address owner***: The VRRP Instance that has the IP address(es) as real interface address(es). This is the VRRP Instance that, when up, will respond to packets addressed to one of these IP address(es) for ICMP, TCP connections, ...
- ***MASTER state***: VRRP Instance state when it is assuming the responsibility of forwarding packets sent to the IP address(es) associated with the VRRP Instance. This state is illustrated on "Case study: Failover" by a red line.
- ***BACKUP state***: VRRP Instance state when it is capable of forwarding packets in the event that the current VRRP Instance MASTER fails.
- ***Real Load balancer***: An LVS director running one or many VRRP Instances.
- ***Virtual Load balancer***: A set of Real Load balancers.
- ***Synchronized Instance***: VRRP Instance with which we want to be synchronized. This provides VRRP Instance monitoring.
- ***Advertisement***: The name of a simple VRRPv2 packet sent to a set of VRRP Instances while in the MASTER state.

III. Software Architecture

3.1 Global View and localization



Our software architecture deals with 4 Linux kernel components:

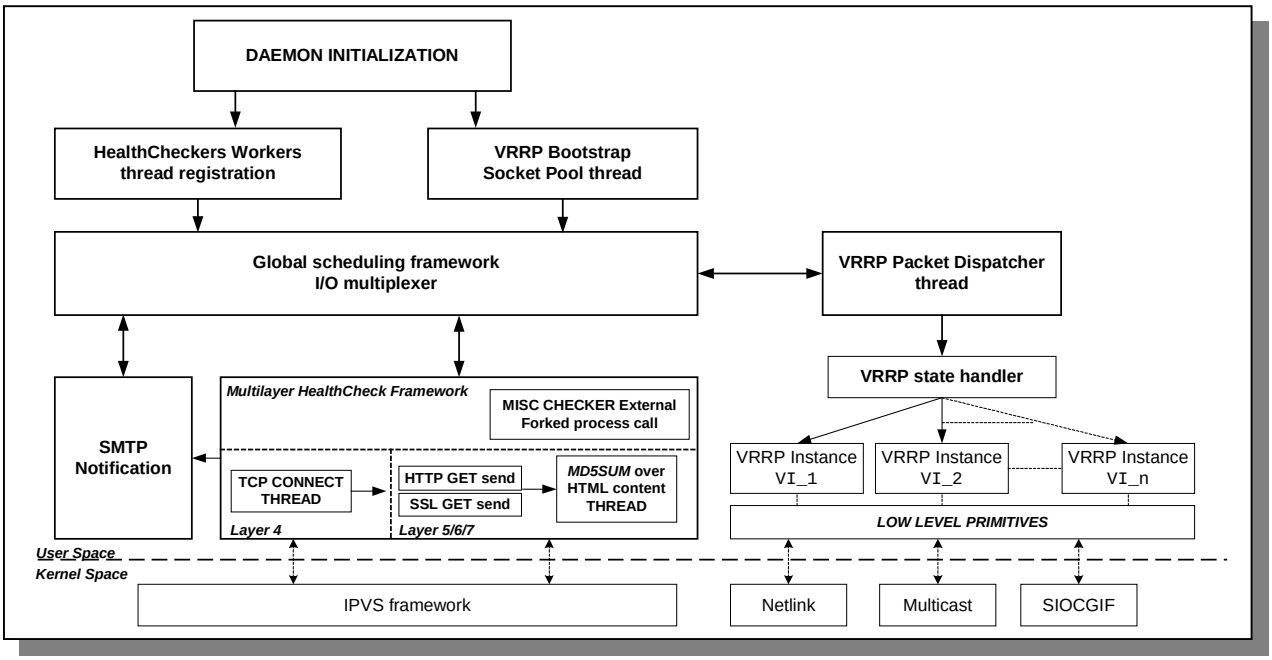
- *LVS kernel framework*: Using the setsockopt call for kernel 2.2 and the setsockopt netfilter call for kernel 2.4.

- *IPCHAINS framework*: For kernel 2.2 in LVS NAT architecture we use an internal IPCHAINS wrapper to send MASQ chains to the kernel. This is only used when running Linux 2.2 kernel. On kernel 2.4 IPVS code handles specific NAT rules using the netfilter call.

- *NETLINK Interface*: For the Hot Standby protocol (VRRP), we use the NETLINK interface to set/remove VRRP VIP.

- *MULTICAST*: For the VRRP part, advertisements are sent to a MULTICAST group.

3.2 Software design



The following figure illustrates the Keepalived internal software implementation components. Keepalived uses a fully multithreaded framework based on a central I/O multiplexer. The 2 main components are:

- *Health-checker worker threads*: Each health-check is registered to our global scheduling framework. These workers perform health-checks using on the Keepalived health-check framework. The health-check frameworks currently implements 3 checkers :
  - *TCP CHECK*: Performing a LAYER3 check.
  - *HTTP GET*: Checking a remote HTTP server html content integrity.
  - *SSL GET*: Checking a remote SSL server html content integrity.
  - *MISC CHECK*: Performing user defined integrity checks.

- *VRRP Packet Dispatcher*: Demultiplexing specific I/O to handle VRRP Instance corresponding.

These 2 main components use the following low-level primitives:

- *SMTP notification*: An SMTP wrapper using asynchronous stream process. This primitive enables Keepalived to send email notifications.

- *IPVS framework*: The LVS kernel interface for real server pool manipulation. All the IPVS load balancing methods are implemented eg: LVS NAT, DR & TUN.

- *Netlink*: Kernel routing interface for the VRRP part. Provides VRRP VIP manipulation.

- *Multicast*: For sending VRRP adverts we use multicast (need to handle multicast binding to specific interface, ...)

- *IPCHAINS framework*: This is only used if running Linux kernel 2.2 to automatically set MASQ chains. This part is obsolete for newer kernel since LVS natively deals with MASQ rules as it is a NETFILTER module. Only implemented for compatibility purpose.

- *SYSLOG*: All daemon notification messages are logged using the syslog daemon.

IV. Healthcheck framework

As described in the global Keepalived software design, the current multilayer health-check framework implements the following checker modules:

- ***TCP_CHECK***: Working at layer4. To ensure this check, we use a TCP Vanilla check using nonblocking/timed-out TCP connections. If the remote server does not reply to this request (timed-out), then the test is wrong and the server is removed from the server pool.

- ***HTTP_GET***: Working at layer5. Performs a GET HTTP to a specified URL. The get result is then summed using the MD5 algorithm. If this sum does not match with the expected value, the test is wrong and the server is removed from the server pool. This module implements a multi-URL get check on the same service. This functionality is useful if you are using a server hosting more than one application server. This functionality gives you the ability to check if an application server is working properly. The MD5 digests are generated using the genhash utility (included in the keepalived package).

- ***SSL_GET***: Same as HTTP_GET but uses a SSL connection to the remote webservers.

- ***MISC_CHECK:*** This check allows a user defined script to be run as the health checker. The result must be 0 or 1. The script is run on the director box and this is an ideal way to test in-house applications. Scripts that can be run without arguments can be called using the full path (i.e. /path_to_script/script.sh). Those requiring arguments need to be enclosed in double quotes (i.e. "/path_to_script/script.sh arg_1 ... arg_n")

The goal for Keepalived is to define a generic framework easily extensible for adding new checkers modules. If you are interested into checkers development, you can read the HackingGuide which exposes the internal software implementation and style guide used (this guide is currently under writing process).

V. Failover framework: VRRP framework

For director failover Keepalived implements the VRRP protocol. To quickly describe this protocol :

> *"VRRP specifies an election protocol that dynamically assigns responsibility for a virtual router to one of the VRRP routers on a LAN. The VRRP router controlling the IP address(es) associated with a virtual router is called the Master, and forwards packets sent to these IP addresses. The election process provides dynamic fail over in the forwarding responsibility should the Master become unavailable. This allows any of the virtual router IP addresses on the LAN to be used as the default first hop router by end-hosts. The advantage gained from using VRRP is a higher availability default path without requiring configuration of dynamic routing or router discovery protocols on every end-host." [rfc2338].*

NB: This framework is LVS independent, so you can use it for LVS director failover, even for other Linux routers needing a Hot-Standby protocol. This framework has been completely integrated in the Keepalived daemon for design & robustness reasons.

The main functionalities provided by this framework are:

- ***Failover***: The native VRRP protocol purpose, based on a roaming set of VRRP VIPs.
- ***VRRP Instance synchronization***: We can specify a state monitoring between 2 VRRP Instances. It guarantees that 2 VRRP Instances remain in the same state. The synchronized instances monitor each other.
- ***Nice Fallback***
- ***Advert Packet integrity***: Using IPSEC-AH ICV.
- ***System call***: During a VRRP state transition, an external script/program can be called.

VI. Installing Keepalived

Before installing Keepalived, any previously installed version should be removed.

1. Download the latest Keepalived source code from the web site and unzip/untar it. In order to compile Keepalived you need to have the following libraries installed:

- ***OpenSSL***, <http://www.openssl.org>: This library is needed for MD5 and SSL support. <Debian – libssl-dev>
- ***popt***, <ftp://ftp.rpm.org/pub/rpm/dist/rpm-4.0.x/> : Used for command line parsing. <Debian – libpopt-dev>

You will also need the Linux kernel source with the ipvs patches if you intend to use Keepalived with LVS.

2. Then simply compile the daemon and the genhash utility.

```
[root@lvs keepalived]# ./configure
[root@lvs keepalived]# make
[root@lvs keepalived]# make install
```

3. All the binary and template configuration file are installed. You may need to create a call to the keepalived daemon in your rc file. If you are using RedHat Linux, an example initialization setup would be:

```
[root@lvs keepalived]# ln -s /etc/rc.d/init.d/keepalived.init /etc/rc.d/rc3.d/S99keepalived
```

If you are running Debian Linux this would be:

```
[root@lvs keepalived]# ln -s /etc/init.d/keepalived.init /etc/rc2.d/S99keepalived
```

Note: The link should be added in your default run level directory.

VII. Keepalived configuration synopsis

The Keepalived configuration file uses the following synopsis (configuration keywords are Bold/Italic):

7.1 Global definitions synopsis

```
global_defs {
   notification_email {
     email
     email
   }
   notification_email_from email
   smtp_server host
   smtp_connect_timeout num
   lvs_id string
}
```

Keyword	Definition	Type
global_defs	identify the global def configuration block	
notification_email	email accounts that will receive the notification mail	List
notification_email_from	email to use when processing “MAIL FROM:” SMTP command	List
smtp_server	remote SMTP server to use for sending mail notifications	alphanum
smtp_connection_timeout	specify a timeout for SMTP stream processing	numerical
lvs_id	specify the name of the LVS director	alphanum

Email type: Is a string using charset as specified into the SMTP RFC eg: “user@domain.com”

7.2 Virtual server definitions synopsis

```
virtual_server (@IP PORT)|(fwmark num) {
    delay_loop num
    lb_algo rr|wrr|lc|wlc|sh|dh|lblc
    lb_kind NAT|DR|TUN
    (nat_mask @IP)
    persistence_timeout num
    persistence_granularity @IP
    virtualhost string
    protocol TCP|UDP

    sorry_server @IP PORT

    real_server @IP PORT {
        weight num
        TCP_CHECK {
            connect_port num
            connect_timeout num
        }
    }
    real_server @IP PORT {
        weight num
        MISC_CHECK {
            misc_path /path_to_script/script.sh
            (or misc_path "/path_to_script/script.sh <arg_list>")
            }
        }
   }
   real_server @IP PORT {
        weight num
        HTTP_GET|SSL_GET {
            url {              # You can add multiple url block
              path alphanum
              digest alphanum
            }
            connect_port num
            connect_timeout num
            nb_get_retry num
            delay_before_retry num
        }
   }
}
```

Keyword	Definition	Type
virtual_server	identify a virtual server definition block	
fwmark	specify that virtual server is a FWMARK	
delay_loop	specify in seconds the interval between checks	numerical
lb_algo	select a specific scheduler (`rr|wrr|lc|wlc`…)	string
lb_kind	select a specific forwarding method (NAT|DR|TUN)	string
persistence_timeout	specify a timeout value for persistent connections	numerical
persistence_granularity	specify a granularity mask for persistent connections	
Virtualhost	specify a HTTP virtualhost to use for HTTP|SSL_GET	alphanum
protocol	specify the protocol kind (TCP|UDP)	numerical
sorry_server	server to be added to the pool if all real servers are down	
real_server	specify a real server member	
Weight	specify the real server weight for load balancing decisions	numerical
TCP_CHECK	check real server availability using TCP connect	
MISC_CHECK	check real server availability using user defined script	
misc_path	identify the script to run with full path	path
HTTP_GET	check real server availability using HTTP GET request	
SSL_GET	check real server availability using SSL GET request	
url	identify a url definition block	
Path	specify the url path	alphanum
Digest	specify the digest for a specific url path	alphanum
connect_port	connect remote server on specified TCP port	numerical
connect_timeout	connect remote server using timeout	numerical
Nb_get_retry	maximum number of retries	numerical
delay_before_retry	delay between two successive retries	numerical

NB: The "***nat_mask***" keyword is obsolete if you are not using LVS with Linux kernel 2.2 series. This flag give you the ability to define the reverse NAT granularity.

NB: Currently, Healthcheck framework, only implements TCP protocol for service monitoring.

NB: Type "path" refers to the full path of the script being called. Note that for scripts requiring arguments the path and arguments must be enclosed in double quotes (").

7.3 VRRP Instance definitions synopsis

```
vrrp_sync_group string {
    group {
        string
        string
    }
    notify_master /path_to_script/script_master.sh
            (or notify_master "/path_to_script/script_master.sh <arg_list>")
    notify_backup /path_to_script/script_backup.sh
            (or notify_backup "/path_to_script/script_backup.sh <arg_list>")
    notify_fault /path_to_script/script_fault.sh
            (or notify_fault "/path_to_script/script_fault.sh <arg_list>")
}

vrrp_instance string {
    state MASTER|BACKUP
    interface string
    mcast_src_ip @IP
    lvs_sync_daemon_interface string
    virtual_router_id num
    priority num
    advert_int num
    smtp_alert
    authentication {
        auth_type PASS|AH
        auth_pass string
    }
    virtual_ipaddress {   # Block limited to 20 IP addresses
        @IP
        @IP
        @IP
    }
    virtual_ipaddress_excluded { # Unlimited IP addresses number
        @IP
        @IP
        @IP
    }
    notify_master /path_to_script/script_master.sh
            (or notify_master "/path_to_script/script_master.sh <arg_list>")
    notify_backup /path_to_script/script_backup.sh
            (or notify_backup "/path_to_script/script_backup.sh <arg_list>")
    notify_fault /path_to_script/script_fault.sh
            (or notify_fault "/path_to_script/script_fault.sh <arg_list>")
}
```

Keyword		Definition	Type
vrrp_instance		identify a VRRP instance definition block	
State		specify the instance state in standard use	
Interface		specify the network interface for the instance to run on	string
mcast_src_ip		specify the src IP address value for VRRP adverts IP header	
lvs_sync_daemon_inteface		specify the network interface for the LVS sync_daemon to run on	string
Virtual_router_id		specify to which VRRP router id the instance belongs	numerical
Priority		specify the instance priority in the VRRP router	numerical
advert_int		specify the advertisement interval in seconds (set to 1)	numerical
smtp_alert		Activate the SMTP notification for MASTER state transition	
authentication		identify a VRRP authentication definition block	
	auth_type	specify which kind of authentication to use (PASS|AH)	
	auth_pass	specify the password string to use	string
virtual_ipaddress		identify a VRRP VIP definition block	
virtual_ipaddress_excluded		identify a VRRP VIP excluded definition block (not protocol VIPs)	
notify_master		specify a shell script to be executed during transition to master state	path
notify_backup		specify a shell script to be executed during transition to backup state	path
notify_fault		specify a shell script to be executed during transition to fault state	path
vrrp_sync_group		Identify the VRRP synchronization instances group	string

Path type: A system path to a script eg: "/usr/local/bin/transit.sh <arg_list>"

VIII. Keepalived programs synopsis

Keepalived package comes with 2 programs.

8.1 keepalived daemon

The related command line arguments are :

```
[root@lvs /root]# keepalived –-help
keepalived Version 0.6.1 (06/13, 2002)
Usage:
  keepalived
  keepalived -n
  keepalived -f keepalived.conf
  keepalived -d
  keepalived -h
  keepalived -v

Commands:
Either long or short options are allowed.
  keepalived --dont-fork       -n       Dont fork the daemon process.
  keepalived --use-file        -f       Use the specified configuration file.
                                        Default is /etc/keepalived/keepalived.conf.
  keepalived --dump-conf       -d       Dump the configuration data.
  keepalived --log-console     -l       Log message to local console.
  keepalived --help            -h       Display this short inlined help screen.
  keepalived --version         -v       Display the version number
```

8.2 genhash utility

All the digest strings are generated with the genhash software. The genhash global synopsis is:

```
[root@lvs /root]# genhash –-help
genhash Version 0.5.8 (05/17, 2001)
Usage:
  genhash -s server-address -p port -u url
  genhash -S -K priv-key-file -P pem-password -s server-address -p port -u url
  genhash -S -K priv-key-file -P pem-password -C cert-file -s server-address -p port -u url
  genhash -h
  genhash -v

Commands:
Either long or short options are allowed.
  genhash --use-ssl         -S       Use SSL connection to remote server.
  genhash --server          -s       Use the specified remote server address.
  genhash --port            -p       Use the specified remote server port.
  genhash --url             -u       Use the specified remote server url.
  genhash --use-private-key -K       Use the specified SSL private key.
  genhash --use-password    -P       Use the specified SSL private key password.
  genhash --use-virtualhost -V       Use the specified VirtualHost GET query.
  genhash --use-certificate -C       Use the specified SSL Certificate file.
  genhash --help            -h       Display this short inlined help screen.
```

8.3 Running Keepalived daemon

1. To run Keepalived simply type:

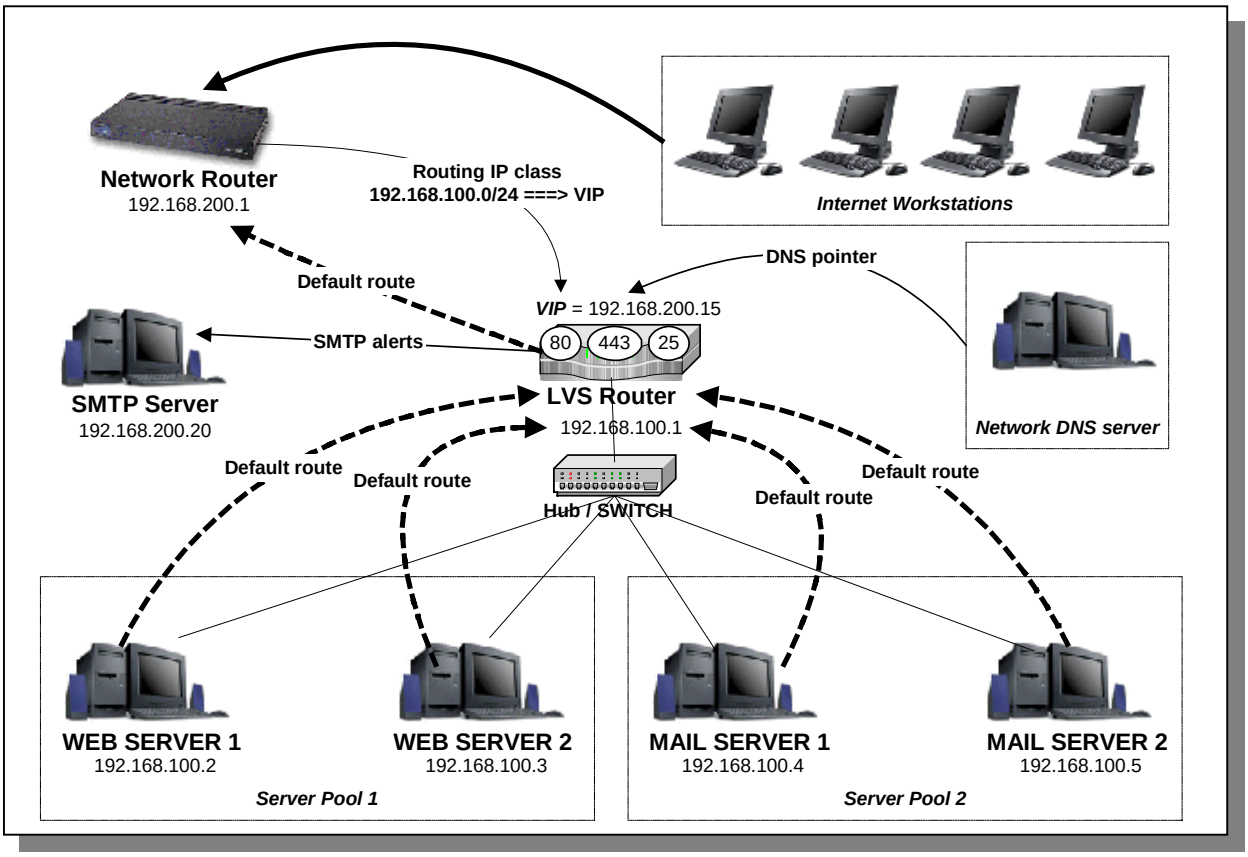
```
[root@lvs tmp]# /etc/rc.d/init.d/keepalived.init start
Starting Keepalived for LVS:                               [  OK  ]
```

2. All daemon messages are logged through the Linux syslog. If you start Keepalived with the “dump configuration data” option, you should see in your `/var/log/messages` (on Debian this may be /var/log/daemon.log depending on your syslog configuration) something like this :

```
Jun  7 18:17:03 lvs1 Keepalived: Starting Keepalived v0.6.1 (06/13, 2002)
Jun  7 18:17:03 lvs1 Keepalived: Configuration is using : 92013 Bytes
Jun  7 18:17:03 lvs1 Keepalived: ------< Global definitions >------
Jun  7 18:17:03 lvs1 Keepalived:  LVS ID = LVS_PROD
Jun  7 18:17:03 lvs1 Keepalived:  Smtp server = 192.168.200.1
Jun  7 18:17:03 lvs1 Keepalived:  Smtp server connection timeout = 30
Jun  7 18:17:03 lvs1 Keepalived:  Email notification from = keepalived@domain.com
Jun  7 18:17:03 lvs1 Keepalived:  Email notification = alert@domain.com
Jun  7 18:17:03 lvs1 Keepalived:  Email notification = 0633556699@domain.com
Jun  7 18:17:03 lvs1 Keepalived: ------< SSL definitions >------
Jun  7 18:17:03 lvs1 Keepalived:  Using autogen SSL context
Jun  7 18:17:03 lvs1 Keepalived: ------< LVS Topology >------
Jun  7 18:17:03 lvs1 Keepalived:  System is compiled with LVS v0.9.8
Jun  7 18:17:03 lvs1 Keepalived:  VIP = 10.10.10.2, VPORT = 80
Jun  7 18:17:03 lvs1 Keepalived:    VirtualHost = www.domain1.com
Jun  7 18:17:03 lvs1 Keepalived:    delay_loop = 6, lb_algo = rr
Jun  7 18:17:03 lvs1 Keepalived:    persistence timeout = 50
Jun  7 18:17:04 lvs1 Keepalived:    persistence granularity = 255.255.240.0
Jun  7 18:17:04 lvs1 Keepalived:    protocol = TCP
Jun  7 18:17:04 lvs1 Keepalived:    lb_kind = NAT
Jun  7 18:17:04 lvs1 Keepalived:    sorry server = 192.168.200.200:80
Jun  7 18:17:04 lvs1 Keepalived:    RIP = 192.168.200.2, RPORT = 80, WEIGHT = 1
Jun  7 18:17:04 lvs1 Keepalived:    RIP = 192.168.200.3, RPORT = 80, WEIGHT = 2
Jun  7 18:17:04 lvs1 Keepalived:  VIP = 10.10.10.3, VPORT = 443
Jun  7 18:17:04 lvs1 Keepalived:    VirtualHost = www.domain2.com
Jun  7 18:17:04 lvs1 Keepalived:    delay_loop = 3, lb_algo = rr
Jun  7 18:17:04 lvs1 Keepalived:    persistence timeout = 50
Jun  7 18:17:04 lvs1 Keepalived:    protocol = TCP
Jun  7 18:17:04 lvs1 Keepalived:    lb_kind = NAT
Jun  7 18:17:04 lvs1 Keepalived:    RIP = 192.168.200.4, RPORT = 443, WEIGHT = 1
Jun  7 18:17:04 lvs1 Keepalived:    RIP = 192.168.200.5, RPORT = 1358, WEIGHT = 1
Jun  7 18:17:05 lvs1 Keepalived: ------< Health checkers >------
Jun  7 18:17:05 lvs1 Keepalived:  192.168.200.2:80
Jun  7 18:17:05 lvs1 Keepalived:    Keepalive method = HTTP_GET
Jun  7 18:17:05 lvs1 Keepalived:    Connection timeout = 3
Jun  7 18:17:05 lvs1 Keepalived:    Nb get retry = 3
Jun  7 18:17:05 lvs1 Keepalived:    Delay before retry = 3
Jun  7 18:17:05 lvs1 Keepalived:    Checked url = /testurl/test.jsp,
Jun  7 18:17:05 lvs1 Keepalived:      digest = 640205b7b0fc66c1ea91c463fac6334d
Jun  7 18:17:05 lvs1 Keepalived:  192.168.200.3:80
Jun  7 18:17:05 lvs1 Keepalived:    Keepalive method = HTTP_GET
Jun  7 18:17:05 lvs1 Keepalived:    Connection timeout = 3
Jun  7 18:17:05 lvs1 Keepalived:    Nb get retry = 3
Jun  7 18:17:05 lvs1 Keepalived:    Delay before retry = 3
Jun  7 18:17:05 lvs1 Keepalived:    Checked url = /testurl/test.jsp,
Jun  7 18:17:05 lvs1 Keepalived:      digest = 640205b7b0fc66c1ea91c463fac6334c
Jun  7 18:17:05 lvs1 Keepalived:    Checked url = /testurl2/test.jsp,
Jun  7 18:17:05 lvs1 Keepalived:      digest = 640205b7b0fc66c1ea91c463fac6334c
Jun  7 18:17:06 lvs1 Keepalived:  192.168.200.4:443
Jun  7 18:17:06 lvs1 Keepalived:    Keepalive method = SSL_GET
Jun  7 18:17:06 lvs1 Keepalived:    Connection timeout = 3
Jun  7 18:17:06 lvs1 Keepalived:    Nb get retry = 3
Jun  7 18:17:06 lvs1 Keepalived:    Delay before retry = 3
Jun  7 18:17:06 lvs1 Keepalived:    Checked url = /testurl/test.jsp,
Jun  7 18:17:05 lvs1 Keepalived:      digest = 640205b7b0fc66c1ea91c463fac6334d
Jun  7 18:17:06 lvs1 Keepalived:    Checked url = /testurl2/test.jsp,
Jun  7 18:17:05 lvs1 Keepalived:      digest = 640205b7b0fc66c1ea91c463fac6334d
Jun  7 18:17:06 lvs1 Keepalived:  192.168.200.5:1358
Jun  7 18:17:06 lvs1 Keepalived:    Keepalive method = TCP_CHECK
Jun  7 18:17:06 lvs1 Keepalived:    Connection timeout = 3
Jun  7 18:17:06 lvs1 Keepalived: Registering Kernel netlink reflector
```

IX. Case Study: Healthcheck

As an example we can introduce the following LVS topology:



First of all you need a well-configured LVS topology. In the rest of this document, we will assume that all system configurations have been done. This kind of topology is generally implemented in a DMZ architecture. For more information on LVS NAT topology and system configuration please read the nice Joseph Mack LVS HOWTO.

9.1 Main architecture components

- *LVS Router*: Owning the load balanced IP Class routed (192.168.100.0/24).
- *Network Router*: The default router for the entire internal network. All the LAN workstations are handled through this IP address.
- *Network DNS Server*: Referencing the internal network IP topology.
- *SMTP Server*: SMTP server receiving the mail alerts.
- SERVER POOL: Set of servers hosting load balanced services.

9.2 Server pool specifications

In this sample configuration we have 2 server pools:

- *Server pool 1:* Hosting the HTTP & SSL services. Each server owns two application servers (IBM WEBSPHERE & BEA WEBLOGIC)
- *Server pool 2*: Hosting the SMTP service.

9.3 Keepalived configuration

You are now ready to configure the Keepalived daemon according to your LVS topology. The whole configuration is done in the **`/etc/keepalived/keepalived.conf`** file. In our case study this file looks like:

```
# Configuration File for keepalived

global_defs {
   notification_email {
     admin@domain.com
     0633225522@domain.com
   }
   notification_email_from keepalived@domain.com
   smtp_server 192.168.200.20
   smtp_connect_timeout 30
   lvs_id LVS_MAIN
}

virtual_server 192.168.200.15 80 {
    delay_loop 30
    lb_algo wrr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP

    sorry_server 192.168.100.100 80

    real_server 192.168.100.2 80 {
        weight 2
        HTTP_GET {
            url {
              path /testurl/test.jsp
              digest ec90a42b99ea9a2f5ecbe213ac9eba03
            }
            url {
              path /testurl2/test.jsp
              digest 640205b7b0fc66c1ea91c463fac6334c
            }
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 2
        }
    }
    real_server 192.168.100.3 80 {
        weight 1
        HTTP_GET {
            url {
              path /testurl/test.jsp
              digest 640205b7b0fc66c1ea91c463fac6334c
            }
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 2
        }
    }
}

virtual_server 192.168.200.15 443 {
    delay_loop 20
    lb_algo rr
    lb_kind NAT
    persistence_timeout 360
    protocol TCP

    real_server 192.168.100.2 443 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
```

```
    real_server 192.168.100.3 443 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
}

virtual_server 192.168.200.15 25 {
    delay_loop 15
    lb_algo wlc
    lb_kind NAT
    persistence_timeout 50
    protocol TCP

    real_server 192.168.100.4 25 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
    real_server 192.168.100.5 25 {
        weight 2
        TCP_CHECK {
            connect_timeout 3
        }
    }
}
```

According to this configuration example, the Keepalived daemon will drive the kernel using following information:

- The LVS server will own the name: LVS_MAIN
- Notification:
  - SMTP server will be: 192.168.200.20
  - SMTP connection timeout is set to: 30 seconded
  - Notification emails will be: admin@domain.com & 0633225522@domain.com
- Load balanced services:
  - HTTP: VIP 192.168.200.15 port 80
    - *Load balancing*: Using Weighted Round Robin scheduler with NAT forwarding. Connection persistence is set to 50 seconds on each TCP service. If you are using Linux kernel 2.2 you need to specify the NAT netmask to define the IPFW masquerade granularity (nat_mask keyword). The delay loop is set to 30 seconds
    - *Sorry Server*: If all real servers are removed from the VS's server pools, we add the sorry_server 192.168.100.100 port 80 to serve clients requests.
    - *Real server* 192.168.100.2 port 80 will be weighted to 2. Failure detection will be based on HTTP_GET over 2 URLS. The service connection timeout will be set to 3 seconds. The real server will be considered down after 3 retries. The daemon will wait for 2 seconds before retrying.
    - *Real server* 192.168.100.3 port 80 will be weighted to 1. Failure detection will be based on HTTP_GET over 1 URL. The service connection timeout will be set to 3 seconds. The real server will be considered down after 3 retries. The daemon will wait for 2 seconds before retrying.

  - SSL : VIP 192.168.200.15 port 443
    - *Load balancing*: Using Round Robin scheduler with NAT forwarding. Connection persistence is set to 360 seconds on each TCP service. The delay loop is set to 20 seconds

- *Real server* 192.168.100.2 port 443 will be weighted to 2. Failure detection will be based on TCP_CHECK. The real server will be considered down after a 3 second connection timeout.
- *Real server* 192.168.100.3 port 443 will be weighted to 2. Failure detection will be based on TCP_CHECK. The real server will be considered down after a 3 second connection timeout.

o SMTP : VIP 192.168.200.15 port 25
- *Load balancing*: Using Weighted Least Connection scheduling algorithm in a NAT topology with connection persistence set to 50 seconds. The delay loop is set to 15 seconds
- *Real server* 192.168.100.4 port 25 will be weighted to 1. Failure detection will be based on TCP_CHECK. The real server will be considered down after a 3 second connection timeout.
- *Real server* 192.168.100.5 port 25 will be weighted to 2. Failure detection will be based on TCP_CHECK. The real server will be considered down after a 3 second connection timeout.

For SSL server health check, we can use SSL_GET checkers. The configuration block for a corresponding real server will look like:

```
 virtual_server 192.168.200.15 443 {
    delay_loop 20
    lb_algo rr
    lb_kind NAT
    persistence_timeout 360
    protocol TCP

    real_server 192.168.100.2 443 {
        weight 1
        SSL_GET {
            url {
              path /testurl/test.jsp
              digest ec90a42b99ea9a2f5ecbe213ac9eba03
            }
            url {
              path /testurl2/test.jsp
              digest 640205b7b0fc66c1ea91c463fac6334c
            }
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 2
        }
    }
    real_server 192.168.100.3 443 {
        weight 1
        SSL_GET {
            url {
              path /testurl/test.jsp
              digest 640205b7b0fc66c1ea91c463fac6334c
            }
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 2
        }
    }
}
```

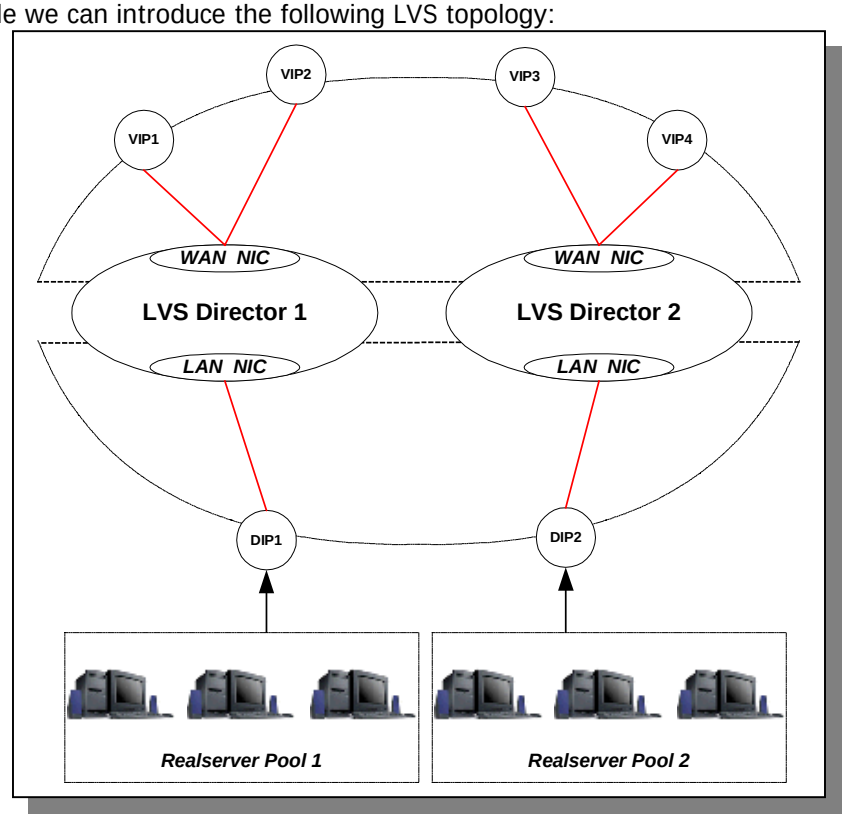
To generate a sum over an URL simply proceed as follows:

```
[root@lvs /root]# genhash –s 192.168.100.2 –p 80 –u /testurl/test.jsp
-------------------------[ HTTP Header Buffer ]-------------------------
0000  48 54 54 50 2f 31 2e 31 - 20 34 30 31 20 55 6e 61   HTTP/1.1 401 Una
0010  75 74 68 6f 72 69 7a 65 - 64 0d 0a 44 61 74 65 3a   uthorized..Date:
0020  20 4d 6f 6e 2c 20 32 33 - 20 41 70 72 20 32 30 30    Mon, 23 Apr 200
0030  31 20 31 35 3a 34 31 3a - 35 34 20 47 4d 54 0d 0a   1 15:41:54 GMT..
0040  41 6c 6c 6f 77 3a 20 47 - 45 54 2c 20 48 45 41 44   Allow: GET, HEAD
0050  0d 0a 53 65 72 76 65 72 - 3a 20 4f 72 61 63 6c 65   ..Server: Oracle
0060  5f 57 65 62 5f 4c 69 73 - 74 65 6e 65 72 2f 34 2e   _Web_Listener/4.
0070  30 2e 38 2e 31 2e 30 45 - 6e 74 65 72 70 72 69 73   0.8.1.0Enterpris
0080  65 45 64 69 74 69 6f 6e - 0d 0a 43 6f 6e 74 65 6e   eEdition..Conten
0090  74 2d 54 79 70 65 3a 20 - 74 65 78 74 2f 68 74 6d   t-Type: text/htm
00a0  6c 0d 0a 43 6f 6e 74 65 - 6e 74 2d 4c 65 6e 67 74   l..Content-Lengt
00b0  68 3a 20 31 36 34 0d 0a - 57 57 57 2d 41 75 74 68   h: 164..WWW-Auth
00c0  65 6e 74 69 63 61 74 65 - 3a 20 42 61 73 69 63 20   enticate: Basic
00d0  72 65 61 6c 6d 3d 22 41 - 43 43 45 53 20 20 20 20   realm="ACCES
00e0  22 0d 0a 43 61 63 68 65 - 2d 43 6f 6e 74 72 6f 6c   "..Cache-Control
00f0  3a 20 70 75 62 6c 69 63 - 0d 0a 0d 0a               : public....
-----------------------------[ HTML Buffer ]-----------------------------
0000  3c 48 54 4d 4c 3e 3c 48 - 45 41 44 3e 3c 54 49 54   <HTML><HEAD><TIT
0010  4c 45 3e 55 6e 61 75 74 - 68 6f 72 69 7a 65 64 3c   LE>Unauthorized<
0020  2f 54 49 54 4c 45 3e 3c - 2f 48 45 41 44 3e 0d 0a   /TITLE></HEAD>..
0030  3c 42 4f 44 59 3e 54 68 - 69 73 20 64 6f 63 75 6d   <BODY>This docum
0040  65 6e 74 20 69 73 20 70 - 72 6f 74 65 63 74 65 64   ent is protected
0050  2e 20 20 59 6f 75 20 6d - 75 73 74 20 73 65 6e 64   .  You must send
0060  0d 0a 74 68 65 20 70 72 - 6f 70 65 72 20 61 75 74   ..the proper aut
0070  68 6f 72 69 7a 61 74 69 - 6f 6e 20 69 6e 66 6f 72   horization infor
0080  6d 61 74 69 6f 6e 20 74 - 6f 20 61 63 63 65 73 73   mation to access
0090  20 69 74 2e 3c 2f 42 4f - 44 59 3e 3c 2f 48 54 4d    it.</BODY></HTM
00a0  4c 3e 0d 0a                -                        L>..
-----------------------[ HTML MD5 final resulting ]-----------------------
MD5 Digest : ec90a42b99ea9a2f5ecbe213ac9eba03
```

The only thing to do is to copy the generated MD5 Digest value generated and paste it into your Keepalived configuration file as a digest value keyword.

X. Case Study: Failover using VRRP

As an example we can introduce the following LVS topology:



10.1 Architecture specification

To create a virtual LVS director using the VRRPv2 protocol, we define the following architecture:

- ***2 LVS directors in active-active configuration.***
- ***4 VRRP Instances per LVS director***: 2 VRRP Instance in the MASTER state and 2 in BACKUP state. We use a symmetric state on each LVS directors.
- ***2 VRRP Instances*** in the same state are to be synchronized to define a persistent virtual routing path.
- ***Strong authentication***: IPSEC-AH is used to protect our VRRP advertisements from spoofed and reply attacks.

The VRRP Instances are compounded with the following IP addresses:

- ***VRRP Instance VI_1***: owning VRRIP VIPs VIP1 & VIP2. This instance defaults to the MASTER state on LVS director 1. It stays synchronized with VI_2.
- ***VRRP Instance VI_2***: owning DIP1. This instance is by default in MASTER state on LVS director 1. It stays synchronized with VI_1.
- ***VRRP Instance VI_3***: owning VRRIP VIPs VIP3 & VIP4. This instance is in default MASTER state on LVS director 2. It stays synchronized with VI_4.
- ***VRRP Instance VI_4***: owning DIP2. This instance is in default MASTER state on LVS director 2. It stays synchronized with VI_3.

10.2 Keepalived configuration

The whole configuration is done in the **`/etc/keepalived/keepalived.conf`** file. In our case study this file on LVS director 1 looks like:

```
vrrp_sync_group VG1 {
  VI_1
  VI_2
}
vrrp_sync_group VG2 {
  VI_3
  VI_4
}
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 150
    advert_int 1
    authentication {
        auth_type AH
        auth_pass k@l!ve1
    }
    virtual_ipaddress {
        192.168.200.10
        192.168.200.11
    }
}
vrrp_instance VI_2 {
    state MASTER
    interface eth1
    virtual_router_id 52
    priority 150
    advert_int 1
    authentication {
        auth_type AH
        auth_pass k@l!ve2
    }
    virtual_ipaddress {
        192.168.100.10
    }
}
```

```
vrrp_instance VI_3 {
    state BACKUP
    interface eth0
    virtual_router_id 53
    priority 100
    advert_int 1
    authentication {
        auth_type AH
        auth_pass k@l!ve3
    }
    virtual_ipaddress {
        192.168.200.12
        192.168.200.13
    }
}

vrrp_instance VI_4 {
    state BACKUP
    interface eth1
    virtual_router_id 54
    priority 100
    advert_int 1
    authentication {
        auth_type AH
        auth_pass k@l!ve4
    }
    virtual_ipaddress {
        192.168.100.11
    }
}
```

Then we define the symmetric configuration file on LVS director 2. This means that VI_3 & VI_4 on LVS director 2 are in MASTER state with a higher priority 150 to start with a stable state. Symmetrically VI_1 & VI_2 on LVS director 2 are in default BACKUP state with lower priority of 100.

This configuration file specifies 2 VRRP Instances per physical NIC. When you run Keepalived on LVS director 1 without running it on LVS director 2, LVS director 1 will own all the VRRP VIP. So if you use the ip utility you may see something like: (On Debian the ip utility is part of iproute)

```
[root@lvs1 tmp]# ip address list
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:00:5e:00:01:10 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.5/24 brd 192.168.200.255 scope global eth0
    inet 192.168.200.10/32 scope global eth0
    inet 192.168.200.11/32 scope global eth0
    inet 192.168.200.12/32 scope global eth0
    inet 192.168.200.13/32 scope global eth0
3: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:00:5e:00:01:32 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.5/24 brd 192.168.201.255 scope global eth1
    inet 192.168.100.10/32 scope global eth1
    inet 192.168.100.11/32 scope global eth1
```

Then simply start Keepalived on the LVS director 2 and you will see :

```
[root@lvs1 tmp]# ip address list
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:00:5e:00:01:10 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.5/24 brd 192.168.200.255 scope global eth0
    inet 192.168.200.10/32 scope global eth0
    inet 192.168.200.11/32 scope global eth0
3: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:00:5e:00:01:32 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.5/24 brd 192.168.201.255 scope global eth1
    inet 192.168.100.10/32 scope global eth1
```

Symmetrically on LVS director 2 you will see:

```
[root@lvs2 tmp]# ip address list
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:00:5e:00:01:10 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.5/24 brd 192.168.200.255 scope global eth0
    inet 192.168.200.12/32 scope global eth0
    inet 192.168.200.13/32 scope global eth0
3: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:00:5e:00:01:32 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.5/24 brd 192.168.201.255 scope global eth1
    inet 192.168.100.11/32 scope global eth1
```

The VRRP VIPs are:

- VIP1 = 192.168.200.10
- VIP2 = 192.168.200.11
- VIP3 = 192.168.200.12
- VIP4 = 192.168.200.13
- DIP1 = 192.168.100.10
- DIP2 = 192.168.100.11

The use of VRRP keyword “sync_instance” imply that we have defined a pair of MASTER VRRP Instance per LVS directors ⇔ (VI_1,VI_2) & (VI_3,VI_4). This means that if eth0 on LVS director 1 fails then VI_1 enters the MASTER state on LVS director 2 so the MASTER Instance distribution on

both directors will be: (VI_2) on director 1 & (VI_1,VI_3,VI_4) on director 2. We use “sync_instance” so VI_2 is forced to BACKUP the state on LVS director 1. The final VRRP MASTER instance distribution will be: (none) on LVS director 1 & (VI_1,VI_2,VI_3,VI_4) on LVS director 2. If eth0 on LVS director 1 became available the distribution will transition back to the initial state.

For more details on this state transition please refer to the “Linux Virtual Server High Availability using VRRPv2” paper (available at http://www.linux-vs.org/~acassen/), which explains the implementation of this functionality.

Using this configuration both LVS directors are active at a time, thus sharing LVS directors for a global director. That way we introduce a virtual LVS director.

NB: This VRRP configuration sample is an illustration for a high availability router (not LVS specific). It can be used for many more common/simple needs.

XI. Case Study: Mixing Healthcheck & Failover

For this example, we use the same topology used in the Failover part. The idea here is to use VRRP VIPs as LVS VIPs. That way we will introduce a High Available LVS director performing LVS real server pool monitoring.

11.1 Keepalived configuration

The whole configuration is done in the **`/etc/keepalived/keepalived.conf`** file. In our case study this file on LVS director 1 looks like:

```
# Configuration File for keepalived

global_defs {
   notification_email {
     admin@domain.com
     0633225522@domain.com
   }
   notification_email_from keepalived@domain.com
   smtp_server 192.168.200.20
   smtp_connect_timeout 30
   lvs_id LVS_MAIN
}

# VRRP Instances definitions

vrrp_sync_group VG1 {
  group {
    VI_1
    VI_2
  }
}

vrrp_sync_group VG2 {
  group {
    VI_3
    VI_4
  }
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 150
    advert_int 1
    authentication {
```

```
        auth_type PASS
        auth_pass k@l!ve1
    }
    virtual_ipaddress {
        192.168.200.10
        192.168.200.11
    }
}

vrrp_instance VI_2 {
    state MASTER
    interface eth1
    virtual_router_id 52
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass k@l!ve2
    }

    virtual_ipaddress {
        192.168.100.10
    }
}

vrrp_instance VI_3 {
    state BACKUP
    interface eth0
    virtual_router_id 53
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass k@l!ve3
    }
    virtual_ipaddress {
        192.168.200.12
        192.168.200.13
    }
}

vrrp_instance VI_4 {
    state BACKUP
    interface eth1
    virtual_router_id 54
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass k@l!ve4
    }
    virtual_ipaddress {
        192.168.100.11
    }
}

# Virtual Servers definitions

virtual_server 192.168.200.10 80 {
    delay_loop 30
    lb_algo wrr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP

    sorry_server 192.168.100.100 80

    real_server 192.168.100.2 80 {
        weight 2
        HTTP_GET {
            url {
              path /testurl/test.jsp
              digest ec90a42b99ea9a2f5ecbe213ac9eba03
            }
```

```
            url {
              path /testurl2/test.jsp
              digest 640205b7b0fc66c1ea91c463fac6334c
            }
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 2
        }
    }

    real_server 192.168.100.3 80 {
        weight 1
        HTTP_GET {
            url {
              path /testurl/test.jsp
              digest 640205b7b0fc66c1ea91c463fac6334c
            }
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 2
        }
    }
}

virtual_server 192.168.200.12 443 {
    delay_loop 20
    lb_algo rr
    lb_kind NAT
    persistence_timeout 360
    protocol TCP

    real_server 192.168.100.2 443 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
    real_server 192.168.100.3 443 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
}
```

We define the symmetric VRRP configuration file on LVS director 2. That way both directors are active at a time, director 1 handling HTTP stream and director 2 SSL stream.