Proxy HTTP transparent avec répartition de charge (LVS Linux)

Cet article présente la solution de proxy transparent clusterisé mise en oeuvre au Centre ERASME. L'usage qui nous intéresse est de fournir un filtrage des sites pornographiques et cela sans nécessiter une configuration particulière sur les postes des usagers.

Le problème c'est qu'au delà d'un certain trafic, un serveur ne suffit plus à filtrer (dans notre cas, au delà de 120 requêtes par seconde sur un P3 800Mhz...). C'est pour cela que la solution est alors de monter un cluster pour que le service soit rendu par plusieurs machines.

Notre objectif étant de pouvoir traiter plus de 200 requêtes par seconde.

Le matériel

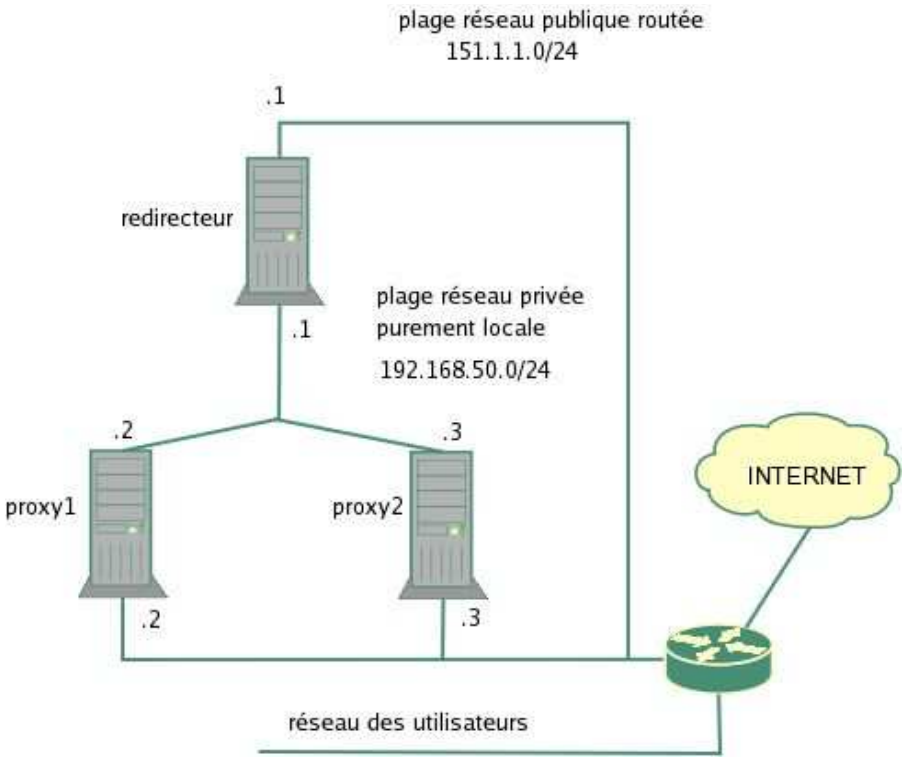
- Cisco Catalyst 6506 (redirection transparente du trafic)
- HP Proliant DL 360 G3, CPU 3GHz, RAM 512Mo, disque 2x18Go 15 RPM RAID1 (répartiteur de charge)
- HP Proliant DL 360 G3, CPU 2.8GHz, RAM 2Go, disque 2x18Go 15 RPM RAID1, disque 2x18Go 15 RPM RAID1 (serveur proxy filtrant)
- HP Proliant DL 360 G3, CPU 2.8GHz, RAM 2Go, disque 2x18Go 15 RPM RAID1, disque 2x18Go 15 RPM RAID1 (serveur proxy filtrant)

Le logiciel

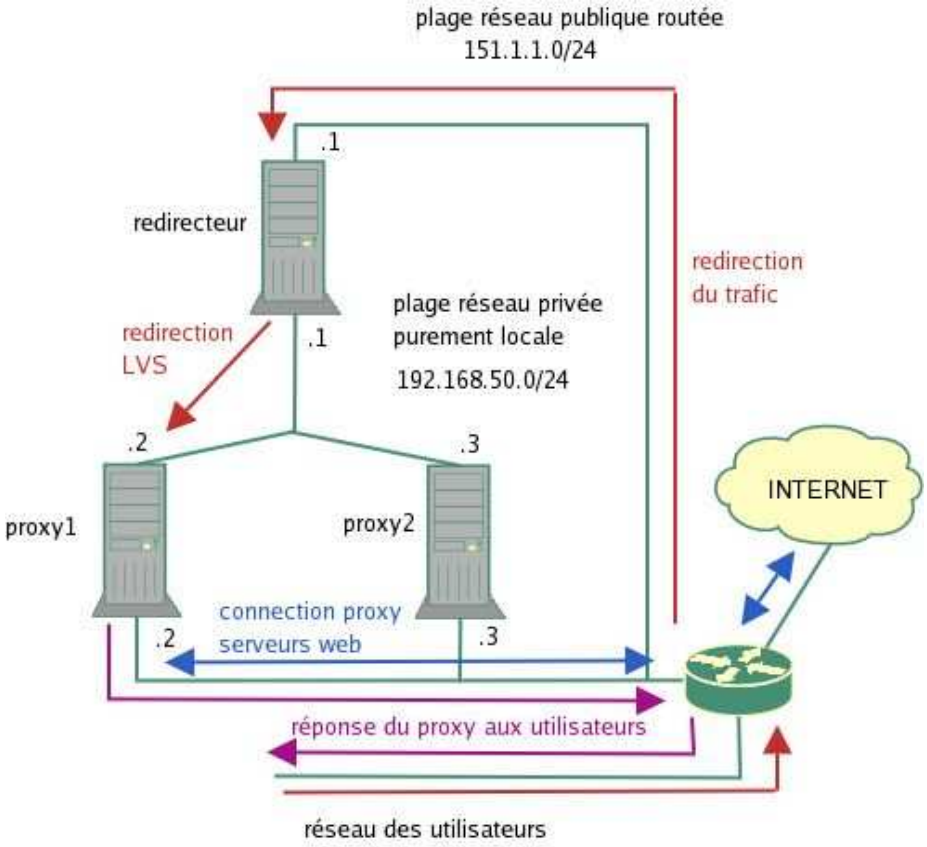
- Fedora Core 2 (distribution Linux)
- LVS (répartition de charge)
- iptables (filtrage, redirection de port, marquage du trafic)
- script perl (pour la surveillance des proxy filtrant)
- Optenet (squid modifié/étendu pour le filtrage sur le contenu...)

La topologie réseau

Schéma réseau du cluster :



Et voici ce que l'on souhaite obtenir lorsqu'un utilisateur se connecte sur un site web :



Configuration du routeur

Dans notre cas, c'est un routeur Cisco qui route le trafic des usagers. Nous utilisons donc le routeur pour rediriger le trafic web (port 80) normalement destiné à Internet sur le redirecteur LVS.

Voici ce que cela donne dans le langage IOS :

```
interface Vlan100
 ip address 201.1.1.1 255.255.255.0
 no ip redirects
 no ip mroute-cache
 ip policy route-map Client_Route_Map
...
access-list 100 deny   tcp any any neq www
access-list 100 permit tcp 201.1.10 0.0.0.255 any
access-list 100 deny   tcp any any
...
route-map Client_Route_Map permit 10
match ip address 100
set ip next-hop 151.1.1.1
```

on peut faire exactement la même chose avec un routeur sous Linux même si la syntaxe change évidemment.

Une fois cela en place, tout le trafic à destination du port 80 se retrouve sur le

redirecteur LVS.

Configuration du redirecteur LVS

Il faut évidemment commencer par installer la distribution Linux, installer **ipvsadm** et configurer correctement les 2 interfaces réseaux.

Ensuite, il faut que la machine fasse du routage. Pour cela, il faut modifier le fichier **/etc/sysctl.conf** :

```
# Kernel sysctl configuration file for Red Hat Linux
#
# For binary values, 0 is disabled, 1 is enabled.  See sysctl(8)
and
# sysctl.conf(5) for more details.

# Controls IP packet forwarding
net.ipv4.ip_forward = 1

# Controls source route verification
net.ipv4.conf.default.rp_filter = 0

# Controls the System Request debugging functionality of the
kernel
kernel.sysrq = 0

# Controls whether core dumps will append the PID to the core
filename.
# Useful for debugging multi-threaded applications.
kernel.core_uses_pid = 1

net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.eth1.send_redirects = 0
net.ipv4.conf.eth0.send_redirects = 0
net.ipv4.conf.lo.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0
```

C’est la ligne **ip_forward = 1** qui permet d’activer le routage. Les lignes **send_redirects = 0** servent à éviter que la machine envoie des ICMP redirect. En effet, cette machine va recevoir des paquets qui ne lui sont pas destinés. Elle risque donc d’envoyer des paquets ICMP redirect pour indiquer que ce n’est pas la bonne route et donner son adresse par défaut.

Pour que cela prenne effet, il faut rebooter la machine ou utiliser la commande **sysctl -p**.

Il faut ensuite "marquer" les paquets qui correspondent au trafic web qui arrive sur le redirecteur. Ce marquage doit nous permettre de traiter le trafic avec LVS. Vous pouvez faire cela avec la commande suivante :

```
iptables -t mangle -A PREROUTING -i eth0 -p tcp -m tcp --dport 80
-j MARK --set-mark 1
iptables -t mangle -A PREROUTING -i eth0 -p tcp -m tcp --dport
8080 -j MARK --set-mark 1
```

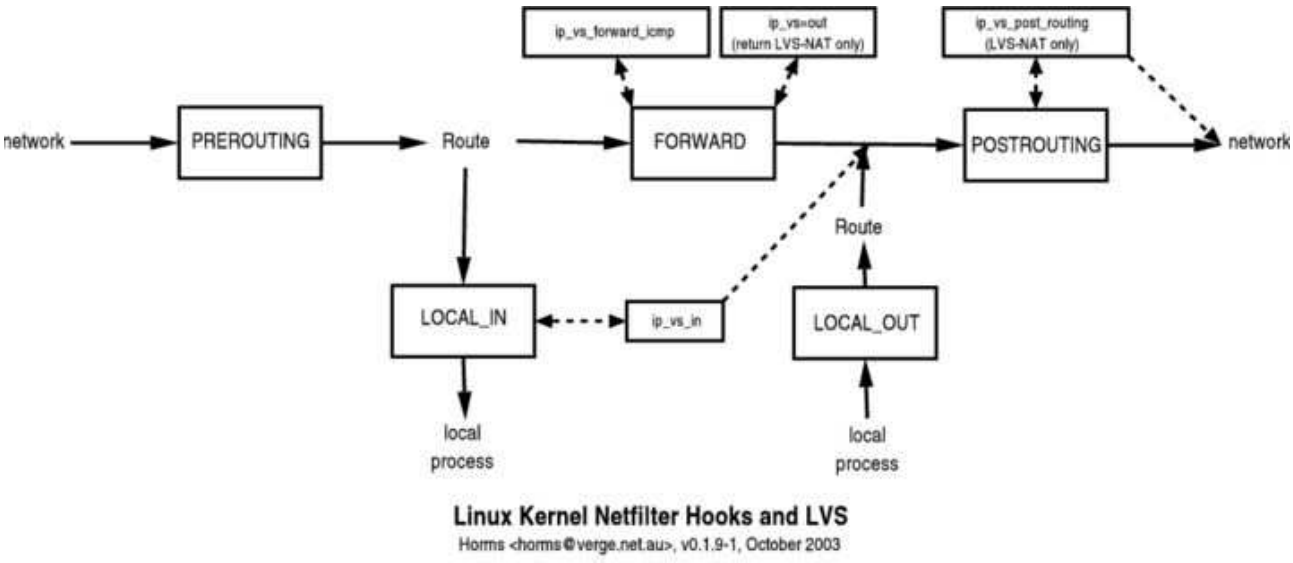
Remplacer eth0 par le nom de l'interface publique du redirecteur (généralement eth0 ou eth1 en fonction de vos branchements).

Ne pas oublier de sauver tout cela dans **/etc/sysconfig/iptables** avec la commande **iptables-save** pour conserver cette règle lors des prochain reboot.

Nous marquons aussi le trafic à destination du port 8080 pour les gens qui configureraient en explicite notre serveur proxy sur le port 8080.

Il faut ensuite router le trafic "marqué" vers l'interface locale. En effet, le trafic n'est pas destinée à la machine mais il faut que LVS traite les paquets. Malheureusement, le routage passe avant LVS dans le kernel. Donc nous routons les paquets localement mais juste pour que cela arrive jusqu'à LVS qui après les fera suivre ailleurs.

Pour information, voici le schéma des traitements réseau dans le kernel :



(tiré de l'article suivant : http://www.austintek.com/LVS/LVS-HO...)

Quoi qu'il en soit, voici les règles de routage à mettre en place pour les paquets marqués :

```
ip rule add prio 100 fwmark 1 table 100
ip route add local 0/0 dev lo table 100
```

il faudra les rajouter quelques part pour les conserver au reboot (comme dans /etc/rc.d/rc.local ou modifier /etc/rc.d/init.d/network).

Maintenant, il faut rentrer les règles pour LVS. Pour cela, le mieux c'est d'éditer/créer le fichier **/etc/sysconfig/ipvsadm** en y entrant les lignes suivantes :

```
-A -f 1 -s wlc
-a -f 1 -r 192.168.50.2:80 -g -w 1
-a -f 1 -r 192.168.50.3:80 -g -w 1
```

La première ligne permet de définir le service en lui indiquant de prendre les paquets marqués "1" et d'utiliser l'algorithme de répartition de charge **wlc** (la liste des choix est disponible avec **man ipvsadm**).

Les deux autres lignes servent à définir les machines qui fournissent véritablement le service. Dans notre cas nous faisons du "Direct Routing" (option -g). Dans cette configuration, les trois machines doivent être sur le même domaine de collision (même plage réseau, ici 192.168.50.0/24). Dans cette configuration, le redirecteur ne fait que modifier les entêtes ethernet des paquets pour remplacer l'adresse MAC de destinataire par celle d'un des serveurs réel et fait suivre. C'est la méthode pas plus performante car le serveur réel va recevoir les paquets, les traiter localement et renvoyer la réponse directement à l'utilisateur sans repasser par le redirecteur. Par ailleurs, la modification de la MAC destination du paquet est très peu coûteuse en CPU par rapport aux autres méthodes possibles (NAT, tunnel).

Une fois les règles définies, il faut (re)démarrer ipvsadm avec la commande **service ipvsadm start**.

ATTENTION, sous la Fedora Core 2, au boot de la machine, les règles ipvsadm sont démarrées avant les règles iptables. Il faut changer cela, sans quoi ipvsadm ne prend pas en compte nos règles.

Configuration des serveurs proxy

Il fautcommencer par installer le proxy HTTP (squid ou optenet). Ensuite, la configuration du serveur est identique à un serveur proxy transparent sans spécificité par rapport à LVS.

A savoir donc que le trafic qui arrive sur la machine arrive a destination des serveurs sur le web (donc pas du tout l'adresse IP du serveur) et sur le port 80 ou 8080 (si conf explicite et donc avec l'IP destination du redirecteur).

Il faut donc configurer IPTables pour faire de la "redirection de port". Nous pouvons faire cela avec les règles suivantes :

```
iptables -t nat -A PREROUTING -p tcp -m tcp --dport 80 -j REDIRECT
--to-ports 8080
iptables -t nat -A PREROUTING -p tcp -m tcp --dport 8080 -j
REDIRECT --to-ports 8080
```

Ne pas oublier de sauver tout cela dans le fichier **/etc/sysconfig/iptables** avec

iptables-save.

Et voilà la machine est prête. Il faut alors en faire une deuxième pour l'autre serveur proxy (ou plus si nécessaire).

Maintenant tout doit fonctionner.

Tolérance de panne

Le problème avec cette solution, c'est que nous avons beaucoup plus de points de défaillance possibles. En effet, si l'un des deux proxy tombe en panne, une requête sur deux n'aboutira pas. Si le redirecteur tombe en panne, plus rien ne fonctionnera.

Dans notre cas, nous avons souhaité pouvoir résister aux pannes des deux serveurs proxy mais pas du redirecteur. Cela permet de faire de la maintenance sur les proxy sans engendrer de gène sur le réseau.

Pour cela, nous avons donc développé un démon sur le redirecteur. Démon qui surveille l'état des serveurs proxy et change à la volée les règles de LVS. Dans le pire des cas, le redirecteur stoppe LVS et démarre un proxy local ne fonctionnant qu'en mémoire.

Vous pouvez récupérer notre script qui est joint à l'article (**squidwatcher**) et le script init.d qui correspond (**squidwatch**).

Pour cette phase, libre à chacun de mettre en place ce qu'il veut. En sachant que supporter la panne du redirecteur n'est pas ce qu'il y a de plus simple mais est tout à fait faisable.

```
#!/usr/bin/perl -w
use strict;
use POSIX;
use IO::Socket;
use Sys::Syslog;

#########################################################################
# squidwatcher
#
# (C) 2004 Daniel Lacroix <dlacroix@erasme.org>
#
# Note:
#   Part of code were copied and adapted from "ldirectord"
#   http://www.vergenet.net/linux/ldirectord/ CVS version 20040922
#
# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License as
# published by the Free Software Foundation; either version 2 of the
# License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
# 02111-1307  USA
#
#########################################################################

=head1 NAME

squidwatcher

Daemon to monitor real squid proxy and control Linux Virtual Server director

=head1 SYNOPSIS

B<squidwatcher> [B<--debug>]

=head1 OPTIONS

B<--debug> Don't start as daemon and activate debugging message on stdout.

=cut


#########################################################################
# Global variable
#########################################################################

# nomber of second beetween each service check (plus check time)
use constant SLEEP_STEP   => 5;
# service port to test to check if service is working
use constant SERVICE_PORT => 80;
# define a web site to test the porn filter
use constant PORN_WEB_SITE => "http://www.sex.com/";
# list of real server to use/test
my $real_server = [
  {
```



```
    'ip'      => '192.168.50.2',
    'status' => 'WORKING', # [WORKING|DEAD]
  },
  {
    'ip'      => '192.168.50.3',
    'status' => 'WORKING', # [WORKING|DEAD]
  },
];

#########################################################################
# Function
#########################################################################

sub daemon
{
  # 'fork()' so the parent can exit, this returns control to the command
  # line or shell invoking your program.  This step is required so that
  # the new process is guaranteed not to be a process group leader. The
  # next step, 'setsid()', fails if you're a process group leader.
  my $status = fork();

  if($status<0) {
    die("Could not fork: $!");
  }
  if($status>0){
    exit(0);
  }

  # setsid()' to become a process group and session group leader. Since a
  # controlling terminal is associated with a session, and this new
  # session has not yet acquired a controlling terminal our process now
  # has no controlling terminal, which is a Good Thing for daemons.
  if(POSIX::setsid()<0){
    die("Could not setsid");
  }

  # fork()' again so the parent, (the session group leader), can exit.
  # This means that we, as a non-session group leader, can never regain a
  # controlling terminal.
  $status = fork();

  if($status<0) {
    die("Could not fork: $!");
  }
  if($status>0){
    exit(0);
  }


  # 'chdir("/")' to ensure that our process doesn't keep any directory in
  # use. Failure to do this could make it so that an administrator
  # couldn't unmount a filesystem, because it was our current directory.
  if(chdir("/")<0){
    die("Could not chdir");
  }

  # 'close()' fds 0, 1, and 2. This releases the standard in, out, and
  # error we inherited from our parent process. We have no way of knowing
  # where these fds might have been redirected to. Note that many daemons
  # use 'sysconf()' to determine the limit '_SC_OPEN_MAX'.  '_SC_OPEN_MAX'
  # tells you the maximun open files/process. Then in a loop, the daemon
  # can close all possible file descriptors. You have to decide if you
```

```
  # need to do this or not.  If you think that there might be
  # file-descriptors open you should close them, since there's a limit on
  # number of concurrent file descriptors.
  close(STDIN);
  close(STDOUT);
  close(STDERR);

  # Establish new open descriptors for stdin, stdout and stderr. Even if
  # you don't plan to use them, it is still a good idea to have them open.
  # The precise handling of these is a matter of taste; if you have a
  # logfile, for example, you might wish to open it as stdout or stderr,
  # and open '/dev/null' as stdin; alternatively, you could open
  # '/dev/console' as stderr and/or stdout, and '/dev/null' as stdin, or
  # any other combination that makes sense for your particular daemon.
  if(open(STDIN, "</dev/null")<0){
    die("Could not open /dev/null");
  }
  if(open(STDOUT, ">>/dev/console")<0){
    die("Could not open /dev/console");
  }
  if(open(STDERR, ">>/dev/console")<0){
    die("Could not open /dev/console");
  }
}

#
# Return:
# 0 : port not open
# 1 : port open
#
sub check_TCP_open
{
  my ($address, $port) = @_;
  my $remote = IO::Socket::INET->new(
    Proto    => "tcp",
    PeerAddr => $address,
    PeerPort => $port,
    Timeout  => 1);
  my $result = 1;
  $result = 0 unless($remote);
  close($remote) if($result);
  return $result;
}

#
# Return:
# 0 : service not filtering
# 1 : service filtering
#
sub check_porn_filter
{
  my ($address, $port) = @_;
  my $remote = IO::Socket::INET->new(
    Proto    => "tcp",
    PeerAddr => $address,
    PeerPort => $port,
    Timeout  => 1);
  my $result = 1;
  return(0) unless($remote);
  print $remote "GET ".PORN_WEB_SITE." HTTP/1.0\n\n";
  my $res = "";
  while(<$remote>) {
    $res .= $_;
  }
  if($res !~ /index\.erasme\.org/) {
    $result = 0;
  }
  close($remote);
  return $result;
}

#
# Return:
# 0 : serveur down
# 1 : serveur up
#
sub check_server
{
  my ($address) = @_;
  system("ping -i 0.2 -W 1 -c 1 $address > /dev/null");
  if($?) {
    return(0);
  }
  else {
    return(1);
  }
}

#
# Start local only service
#
sub start_local_proxy
{
  system("service optenet start");
  system("iptables -t nat -A PREROUTING -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 8080");
  system("iptables -t nat -A PREROUTING -p tcp -m tcp --dport 3128 -j REDIRECT --to-ports 8080");
  system("iptables -t nat -A PREROUTING -p tcp -m tcp --dport 9090 -j REDIRECT --to-ports 8080");
}

#
# Stop local only service
#
sub stop_local_proxy
{
  system("iptables -t nat -D PREROUTING -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 8080");
  system("iptables -t nat -D PREROUTING -p tcp -m tcp --dport 3128 -j REDIRECT --to-ports 8080");
  system("iptables -t nat -D PREROUTING -p tcp -m tcp --dport 9090 -j REDIRECT --to-ports 8080");
  system("service optenet stop");
}

##############################################################################
# Main program
##############################################################################

my $debug = 0;

if(join(' ', @ARGV) =~ '--debug') {
  $debug = 1;
}

# become a daemon
&daemon() unless($debug);

# open syslog facility
```

```
openlog('squidwatcher', 'pid', 'daemon');
syslog('info', 'squid LVS watcher service started');

# 1 if all server are down
my $minimum_service = 0;

# check for ever
while(1) {
  my $state_changed = 0;
  # check working server
  my @working_server;
  $#working_server = -1;
  print "New service check round:\n" if($debug);
  foreach my $server (@{$real_server}) {
    if(&check_porn_filter($server->{'ip'}, SERVICE_PORT)) {
      if($server->{'status'} eq 'DEAD') {
        $state_changed = 1;
        $server->{'status'} = 'WORKING';
      }
      push(@working_server, $server->{'ip'});
      print "WORKING: $server->{ip}\n" if($debug);
    }
    else {
      if($server->{'status'} eq 'WORKING') {
        $state_changed = 1;
        $server->{'status'} = 'DEAD';
        print "DEAD: $server->{ip}\n" if($debug);
      }
    }
  }
  # if the working status has changed, update the setup
  # accordingly
  if($state_changed) {
    print "global status changed\n" if($debug);
    # clear all real server
    system("ipvsadm -C; ipvsadm -A -f 1 -s wlc");
    # if some server are working generate corresponding LVS setup
    if($#working_server != -1) {
      # add each working server to the ipvsadm list
      foreach my $workserver (@working_server) {
        print "ipvsadm -a -f 1 -r $workserver:".SERVICE_PORT." -g -w 1\n" if($debug);
        system("ipvsadm -a -f 1 -r $workserver:".SERVICE_PORT." -g -w 1");
      }
      # log status changer
      syslog('warning', 'status changed, working server list: '.join(',',@working_server));

      # if we were at minimum service stop it now
      if($minimum_service) {
        print "minimum service stop\n" if($debug);
        &stop_local_proxy();
        $minimum_service = 0;
      }
    }
    # oups we are alone working, start a proxy on the redirector
    else {
      print "minimum service start\n" if($debug);
      &start_local_proxy();
      $minimum_service = 1;
      # log status changer
      syslog('warning', 'NO MORE REAL SERVER WORKING MINIMAL MODE');
    }
  }
```



```
  # sleep a litle before next service check
  sleep(SLEEP_STEP);
}

closelog();

exit(0);
```

```
#!/bin/bash
#
# squidwatch        This starts and stops squidwatcher.
#
# chkconfig: 345 99 1
# description: squidwatcher is a daemon to monitor real squid proxy\
#              in the LVS loadbalancing system.
#
# pidfile: /var/run/squidwatcher.pid

PATH=/sbin:/bin:/usr/bin:/usr/sbin:/usr/local/bin

# Source function library.
. /etc/init.d/functions

# Check that we are root ... so non-root users stop here
[ 'id -u' = 0 ] || exit 1

RETVAL=0

prog="squidwatcher"

start(){
    echo -n $"Starting $prog: "
    /usr/local/bin/squidwatcher
    RETVAL=$?
    echo
    return $RETVAL
}

stop(){
    echo -n $"Stopping $prog: "
    killproc $prog
    RETVAL=$?
    echo
    return $RETVAL

}

restart(){
    stop
    start
}

# See how we were called.
case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    status)
        status $prog
        ;;
    restart)
        restart
        ;;
    reload)
        reload
        ;;
    condrestart)
```



```
        condrestart
        ;;
    *)
        echo $"Usage: $0 {start|stop|status|restart}"
        RETVAL=1
esac

exit $RETVAL
```