Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 53(67), Fascicola 2, 2008

Simulating a Load Balancing Implementation on Multiple Default Routes via Different ISPs

Vâtca Dan Stefan, Mocofan Muguras[1]

Abstract – This paper shows a method of implementing and simulating a network in which traffic from an internal network is balanced across two Internet Service Providers to achieve higher service availability and also provide failover in case of lack of service from one provider. This method is very cost effective and also demonstrates a way of utilizing unused backup Internet connection lines.

Keywords: load balancing, failover, network simulation

I. INTRODUCTION

From the past couple of years companies rely more and more on highly available broadband connections to the Internet. This is because the majority of businesses are either engaging in online commerce or are using extensively the Internet for research. In this highly connected world an *always-on* connection to the world is a must.
A possible solution to satisfy this necessity is employing the use of two or more different Internet Service Providers. This way they can use one Internet connection as primary, and the second one as a backup line. Contracting and paying two ISPs for connectivity will prompt the companies to search for a solution to automatically switch all traffic to the *healthy* provider in case one has failed, and also to use both lines for regular traffic if both of them are operating properly.
There exist both commercial and open solutions to this problem. In this paper I will explain how to use VMWare Fusion (www.vmware.com) to simulate a scenario of this type and evaluate various an example configuration.

II. IMPLEMENTATION SCENARIO

I will start by describing the rather simple network layout of this fictitious company that has two service providers called ISP1 and ISP2. Each provider has its own router on a different network. In fig. 1 I depict the layout of the network we are going to simulate.

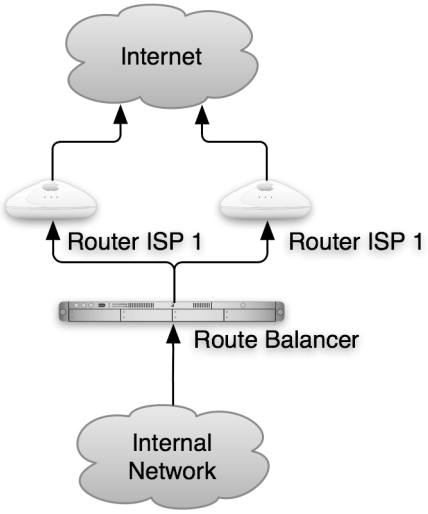

Fig. 1 Network Layout Scenario

In the simulation I will use router software based on Linux from a free router implementation called Syneto. I chose this implementation firstly because I had a great contribution to its architecture, and secondly because it is very small (40MB for the complete system).

III. VIRTUAL NETWORK

The scenario seems rather simple because the figure above implies a lot of components that are not actually depicted. To be able to properly simulate this situation we have to lay down a more complex plan to account for all the missing components.
In the simulation we will be able to use three kinds of devices: switches, routers and network cables.
Switches are simulated by means of virtual network interfaces called *vmnet* by the software we are using. Each *vmnet* behaves like a layer 2 switch or hub that we will use to virtually connect all the virtual machines.
Routers and hosts will be simulated by virtual machines – actually x86 machines – running Linux.
Finally, cables are – not so obviously – simulated by connecting or disconnecting a virtual network

[1] Facultatea de Electronică şi Telecomunicaţii, Departamentul Comunicaţii Bd. V. Pârvan Nr. 2, 300223 Timişoara, e-mail: muguras.mocofan@etc.upt.ro, dan@syneto.net

interface card from a virtual switch. In Fig. 2 I am extending the simple network above including all necessary components for the simulation.

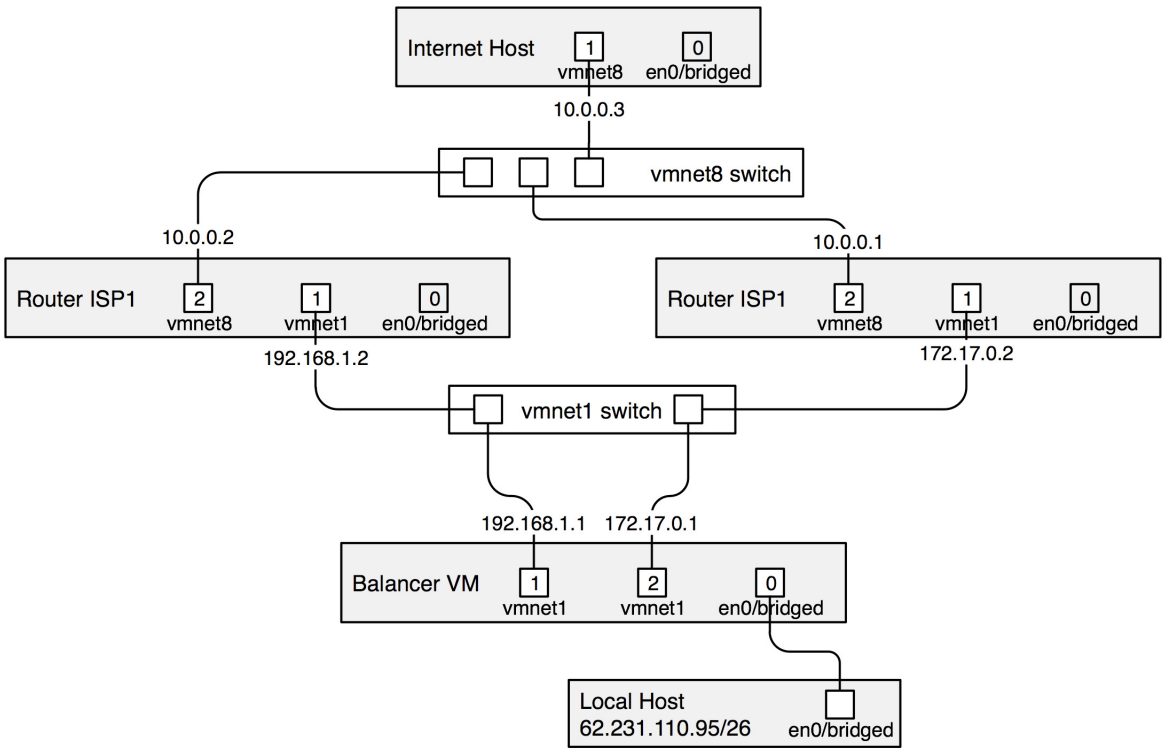

Fig. 2 Network detailed for simulation

The grayed out boxes represent virtual machines running Linux. The boxes inside represent Network Interface Cards (NICs), while the numbers inside the boxes represent the interface number (e.g. 0 for eth0). Grayed out NICs are disabled during simulation, but they are used only during the virtual machine configuration phase.

White boxes represent two virtual switches I used: vmnet1 and vmnet8. I chose to use these two virtual switches because they come preconfigured and on the MacOS X platform there is no easy way to configure additional virtual switches. On the VMWare Server, which runs on both Windows and Linux, configuration of up to 6 additional virtual switches is done easily using the *vmware-config.pl* script.

After planning the virtual network layout we have to create the virtual machines. Each virtual machine will need: 1 virtual CPU, 64 MB of RAM memory, depicted number of virtual network cards, one 64MB virtual hard disk and one virtual CD-ROM drive used to install the operating system.

After every virtual machine is installed started up, we will configure each machine using the IP addresses shown in Fig. 2. On Syneto machines configuration may be done using either the HTTPS interface or from the serial or video console using *config* commands.

The balancer and the routers will also have configured network address translation in order to avoid having to worry about return routes.

I will show an example of configuration steps needed for the balancer virtual machine.

1. First we will use the console to configure IP addresses:

config addr edit eth0 62.231.110.65/26
config addr edit eth1 192.168.1.1/24
config addr edit eth2 172.17.0.1/24

2. Next we need to configure the two gateways and balance traffic across them.

config gateway edit 192.168.1.2 eth1
config gateway add 172.17.0.2

3. Finally, all we have left to do is enable link failover specifying a 2 second interval between link verifications.

config gateway failover 2

We may now verify our gateway balancing and failover configuration using *config gateway show*:

Default gateway [1]: 192.168.1.2 (eth1) (enabled) weight 1 src 192.168.1.1
Default gateway [2]: 172.17.0.2 (eth2) (enabled) weight 1 src 172.17.0.1
Failover: (link monitor) (every 2s)

The core operating system running in the virtual machine is actually Linux. Syneto OS shields the network administrator from the details required to setup the route balancing and failover. While the commands required to setup route balancing are specifying little information, they are just enough for

the configuration programs to setup the Linux kernel to correctly route packets.

IV. HOW IT WORKS

To implement balancing and failover we will need to solve three problems:

1. Routing packets back on the same interface they were sent out from
2. Balancing traffic on the two network interfaces
3. Monitoring route destination – in this case the default route – for reach-ability via any of the two interfaces

To correctly split traffic across the two network interfaces, we have to enable two Linux kernel options: *IP: advanced router* and *IP: policy routing*. These options allow usage of multiple alternative routing tables based on routing rules specified by the administrator.

Using source network address translation (SNAT) can solve the problem of packets returning on the same network interface. In our virtual network setup we will configure network address translation to change the packet source on all *core* virtual machines: Balancer VM and the two ISP Routers.

The ISP Routers will employ only one SNAT rule that will rewrite the source of every packet exiting its network interface connected to *vmnet8* so that all packets seem to have originated on the router itself. This way the Internet clients will know to what router to return packets. When the packets return, the Linux router will use the connection-tracking table to recreate correct response packets.

The SNAT rule will specify that any packet that is supposed to go out thrugh the *vmnet8* switch (i.e. eth2 Ethernet NIC in the case of the ISP routers) will have the source rewritten using 10.0.0.1 or 10.0.0.2 respectively.
On the other hand, the Balancer VM will have to configure two SNAT rules, one for every connection to each ISP. But getting to the rules, let's take a look at how packets traverse the Linux kernel in Fig. 3.

Here we are interested in two processes:

1. Routing decision
2. Postrouting processing

The routing decision is responsible for balancing routes by associating every pair (source, destination) to one of the configured gateways. Specifying a route weight configures how often a certain ISP (gateway) is chosen. Once chosen a route is kept in place for a predefined amount of time before getting replaced.
This means that the balancer does not have very good performances for a small number of internal clients. But as client number increases traffic will more evenly be distributed.

After the routing process takes the decision to which ISP to handle a certain connection, the packets will be inspected and modified by the post-routing process. This process is responsible for changing the source of the packets to match the interface chosen by the routing process.

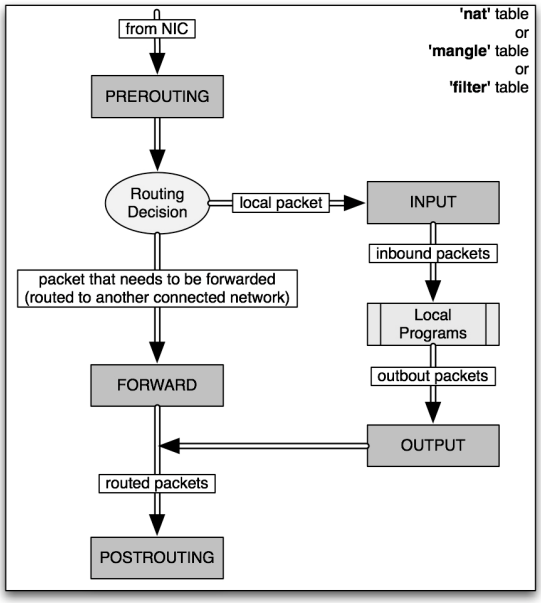

Fig. 3 Linux kernel packet traversal

The config gateway add command will setup route balancing by adding couple of routing rules and their corresponding routing tables. These are the routing rules automatically added:

0: from all lookup local
240: from all lookup main
241: from 192.168.1.0/24 lookup 241
242: from 172.17.0.0/24 lookup 242
243: from all lookup 243
32766: from all lookup main
32767: from all lookup 253

Routing rules 241 and 242 specify rules that apply to the directly connected networks to the provider. These are needed to enforce packets originating from these networks not to be balanced, but ramain local to te respective networks. This way we avoid routing packets destined to ISP1 via ISP2.

The 243 routing rule is actually the rule that specifies the load balancing route. This is the routing table contents:

$ ip route show table 243
default proto static
nexthop via 192.168.1.2 dev eth1 weight 1
nexthop via 172.17.0.2 dev eth2 weight 1

V. FAILOVER

Setting up just the routes will only balance traffic on the two network interfaces. The problem is that once an ISP stops routing traffic correctly the Linux kernel will keep using that gateway, and some connections will be dropped.

Our solution will be to implement failover by writing a daemon (or service) that will run on the Balancer VM to monitor the "health" of the connections.
Defining "health" of the connection is the job of the system administrator. He may choose to monitor the link to its ISPs at more OSI layers:

1. At data link layer: verify network cards' link status
2. At network layer: verify connectivity at layer 3 using ICMP echo-request and echo-reply
3. At transport layer: attempt connections on TCP or UDP on a certain port.
4. At application layer: monitor availability o of a certain URL on a web server

On Syneto OS a Perl script implements failover by verifying the destination specified by the administrator. In our example we specified that the failover daemon should monitor link status of the connected network interfaces every 2 seconds. We could as well specify an IP address like 10.0.0.3 (our Internet Host) or even a TCP port on that host.

The route failover daemon is responsible of maintaining in this situation the 243 routing table adding gateways as they become available or removing them when the failover probes show a gateway as being unavailable.

VI. SIMULATION

Running the simulation means starting all 4 virtual machines as previously configured and trying to connect from the local host to the simulated Internet Host. The only prerequisite of the local host is to have either the default route through the Balancer VM or just a static route to our simulated Internet – network 10.0.0.0/24.
Before starting the simulation, we have to disconnect the virtual cables used to configure each virtual machine (ports marked as gray in Fig. 2). This way the local host can no longer access Internet Host directly, but only via Balancer VM and one of the two ISP routers.
For testing one may use console network connectivity tests like *ping* or *wget* – tools usually available for free can be used to test connectivity at layer 3 or 7 respectively.

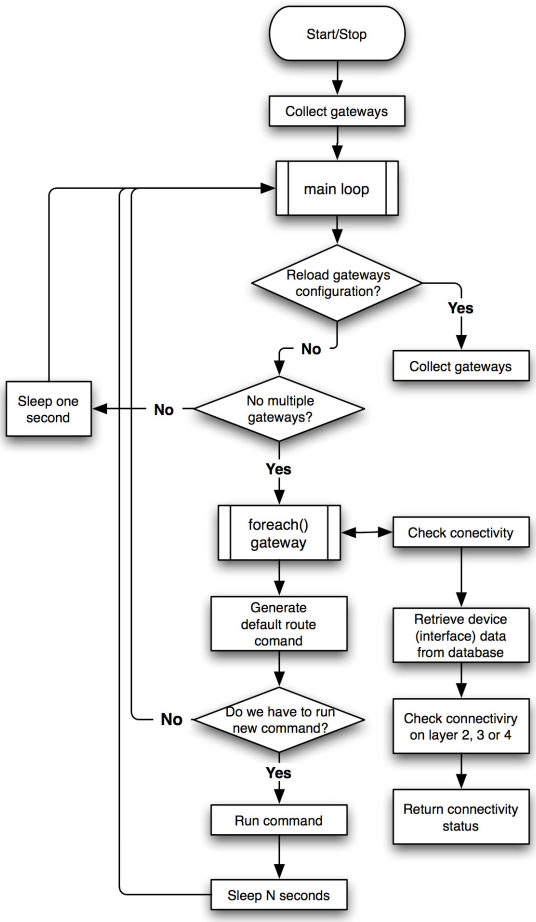

Fig. 4 Failover monitor process flowchart

Now it is time to simulate failures by disconnecting cables from one ISP, and watching what route packets travel. Using the *tcpdump* network analysis tool, we can observe packets coming from ISP1. If we disconnect the virtual cable connecting to ISP1 and connect only the cable leading to ISP2, we will be able to observe packets changing origin to ISP2.

VII. CONCLUSION

This method can be used to verify the feasibility of complex network scenarios as the one presented without disrupting normal network operations. Actually it provides a way to test complex network scenarios without even requiring a network connection.

REFERENCES

[1] Bourke, T., "Server Load Balancing", *O'Reilly & Associates*, 2001.
[2] Hubert B., *Linux Advanced Routing and Traffic Control HOWTO*, http://www.lartc.org, 2002.
[3] http://www.syneto.net/public/products/software_solutions/
[3] http://www.vmware.com/products/fusion/