Netkit
The Networking Sandbox

Networking is very interesting and I'm sure many of us would like to experiment with it, but the unavailability of laboratories with sufficient computers is a stumbling block. What if I tell you that you can turn any computer into a network device at no extra cost?

Netkit is a virtualised environment on the top of User Mode Linux that gives wings to your network experimentation dreams. This article focuses on how to get started with Netkit, including modelling and testing complex networks with it.

In order to master networking we need to hack on real-time network devices such as routers, gateways and other costlier devices that are not always readily available. It is a physical strain plugging in the cables, switching on the devices, logging in to routers, etc, apart from the logical exercise of designing and setting up the network. Anyone attempting all this has to be hands-on with several devices, several interfaces, protocols, physical interconnections, etc.

Netkit, on the other hand, is the laziest and easiest way to learn, understand and test a network on a virtual environment. It is also a great teaching aid.

Using the Netkit toolkit, we can create as many virtual machines as we need. They will be displayed as terminals. Each of the terminals can be programmed using different *nix tools to make them work as different network devices such as routers, a gateway, etc.

What can Netkit do?

It is never advisable to conduct experiments on the primary network of an entity, since it hosts services that are critical for its operations. On the other hand, network equipment could be expensive, and even for simple experiments, sufficient equipment should be available in the same test bed.

Under the circumstances, Netkit—a system to emulate computer networks by creating a real-time environment—comes to the rescue. The one-line description of Netkit is: "The poor man's system for experimenting." It has been developed and maintained by the Network Research Group of the Roma Tre University (in Rome, Italy).

The virtualised environment for the Netkit toolkit is based on UML (User Mode

Linux—*user-mode-linux.sourceforge.net*). Each emulated network device is a virtual GNU/Linux system. Since GNU/Linux is shipped with software supporting most of the network protocols, any machine can be configured to act as a bridge/switch or as a router. Each virtual machine has a console (terminal window) with a root shell.

Installing and setting up Netkit

The hardware requirements for Netkit are pretty basic:

- ≥ 600 MHz CPU
- ~10 MB memory for each VM (depending on the VM configuration)
- ~600 MB of disk space + ~1-20 MB for each VM (depending on the usage of the VM)

As for software requirements, you'll need a GNU/Linux installation and of course, you can download Netkit from *www.netkit.org* if it's not available in your distro's online software repositories. [Netkit works fine on many distributions—see *wiki.netkit.org/index.php/Status*].

Download the following three files: *netkit-X.Y.tar.bz2*, *netkit-filesystem-FX.Y.tar.bz2* and *netkit-kernel-KX.Y.tar.bz2*.

Unpack them in the same location:

```
slynux@gnubox:~$ mkdir ~/netkit
// Follow the following order in extraction of archives
slynux@gnubox:~$  tar -xjf netkit-X.Y.tar.bz2 -C ~/netkit
slynux@gnubox:~$  tar -xjf netkit-filesystem-FX.Y.tar.bz2 ~/netkit
slynux@gnubox:~$  tar -xjf netkit-kernel-KX.Y.tar.bz2 ~/netkit
```

Now it requires you to set up some environment variables to make Netkit work. Append the following lines to your *~/.bashrc*:

```
 export NETKIT_HOME=~/netkit
 export PATH=$PATH:$NETKIT_HOME/bin
 export MANPATH=:$NETKIT_HOME/man
```

You can now check your Netkit installation by opening a new terminal running the following *check_configuration.sh* bash script.

```
slynux@gnubox:~$  cd $NETKIT_HOME
slynux@gnubox:~$  ./check_configuration.sh
```

If it succeeds, your Netkit installation is successful.

An introduction to Netkit commands

We will now look into some of the Netkit commands used to start, configure and monitor Netkit virtual machines and the virtual labs.

- *vstart* starts a new virtual machine
- *vlist* lists the currently running virtual machines
- *vconfig* attaches network interfaces to running VMs
- *vhalt* gracefully halts a virtual machine
- *vcrash* causes a virtual machine to crash
- *vclean* is the "panic command" to clean up all Netkit processes (including VMs) and configuration settings on the host machine

The Netkit lab is an interesting facility that comes inbuilt with the Netkit toolkit. While emulating a large network, it is harder to configure each of the VMs individually, every time you want to conduct a new experiment. Netkit lab facilities help in setting up complex labs consisting of several virtual machines. Here are some commands to manipulate and keep track of the running labs:

- *lstart* starts a Netkit lab
- *lhalt* gracefully halts all the VMs of a lab
- *lcrash* causes all the VMs of a lab to crash
- *lclean* removes temporary files from a lab directory
- *linfo* provides information about a lab without starting it
- *ltest* allows you to run tests to check that the lab is working properly

Starting virtual hosts

Each of the VMs represents a network device or a host. In order to start a virtual machine with the name 'pc1', use the following command:

```
slynux@gnubox:~$ vstart pc1
============= Starting virtual machine "pc1" =============
   Kernel:     /home/slynux/netkit/kernel/netkit-kernel
   Modules:    /home/slynux/netkit/kernel/modules
   Memory:     16 MB
   Model fs:   /home/slynux/netkit/fs/netkit-fs
   Filesystem: /home/slynux/pc1.disk
   Hostfs at:  /home/slynux

Running ==> /home/slynux/netkit/kernel/netkit-kernel modules=/home/
slynux/netkit/kernel/modules name=pc1 title=pc1 umid=pc1 mem=20M
ubd0=/home/slynux/pc1.disk,/home/slynux/netkit/fs/netkit-fs root=98:1
uml_dir=/home/slynux/.netkit/mconsole hosthome=/home/slynux quiet
con0=xterm con1=null SELINUX_INIT=0
```

A console terminal corresponding to pc1 will pop up. You can create any number of virtual hosts using the *vstart* command. We will use options like—*eth0*,—*eth1* to specify the network interface and its connection. For example:

```
slynux@gnubox:~$ vstart pc1 --eth0=A

slynux@gnubox:~$ vstart pc2 --eth0=A
```

Here we have two machines pc1 and pc2. The—*eth0* argument corresponds to the network interface, while 'A' in the—*eth0=A* argument specifies the collision domain of the network—i.e., in the above example, pc1 and pc2 are connected on the same network family as if they are interconnected through a network hub.

If there is a pc3—*vstart pc3—eth0=B*—it means pc3 is connected to a different network hub and pc3 is not reachable from pc1 or pc2. In order to bridge both hubs, we require a router in between.

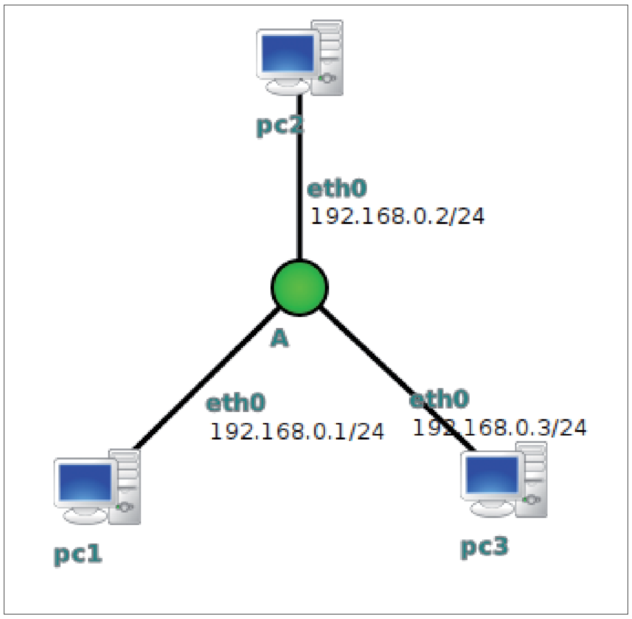

Figure 1: A network with three hosts interconnected via a network hub

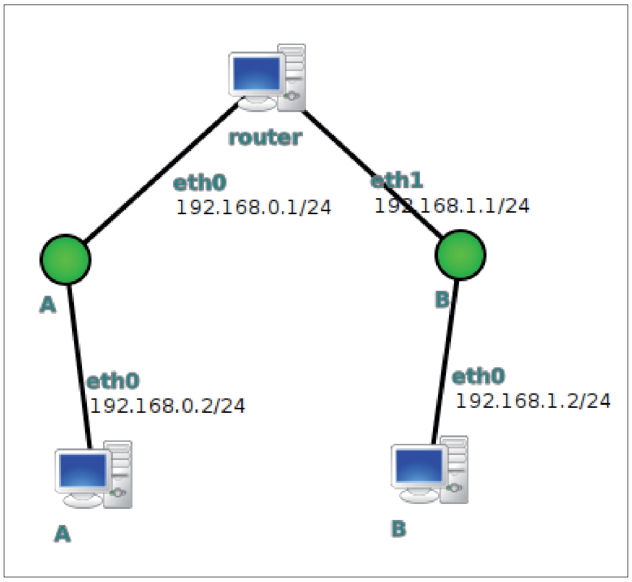

Figure 2: A network with a router to bridge two families of networks

We can shut down the machine from outside the virtual terminal using the command *vhalt [virtual host name]*. If something goes wrong and needs the virtual machine to be forcefully shut down, use the command *vcrash [hostname]*.

The—*con0* argument for the *vstart* command will help you in specifying the terminal to be used for a VM. To use the same terminal in which commands are typed, use—*con0=this* or if you want Konsole as the terminal, use—*con0=konsole*.

Some network manipulation/config commands

- The *ping* command is to check connectivity. We can ping another machine on a network using its IP address/hostname and check the connectivity by looking at the command output. For example: *ping google.com*
- *ifconfig* is used to set the IP address for an interface. To set the IP address of eth0, just use *ifconfig eth0 <ip address> netmask <netmask>*
- The *route* command can be used to set the routes for an IP network. For example, *route add default gw 192.168.0.1 dev eth0* sets 192.168.0.1 as the default gateway for the interface as eth0.
- You can use the *tcpdump* command for packet sniffing. With this command, you can monitor the network data packets transfer through a network interface. For example, *tcpdump -i eth0*
- You can use *traceroute* to trace the packet route. This means that you can find out the gateways through which a packet travels to reach its destination. For example, *traceroute mec.ac.in*

Hands-on network emulation

We will now look at how to emulate simple networks using Netkit. A network with three hosts interconnected via a network hub is shown in Figure 1.

It is very easy to set up a network of machines connected through a common network hub. Execute the following commands:

```
slynux@gnubox:~$ vstart pc1 --eth0=A
slynux@gnubox:~$ vstart pc2 --eth0=A
slynux@gnubox:~$ vstart pc3 --eth0=A
```

These three commands will create three terminals, namely Virtual Console #0 (pc1), Virtual Console #0 (pc2), Virtual Console #0 (pc3).

Let's list the details of active virtual machines using the command, *vlist*.

```
slynux@gnubox:~$ vlist
USER          VHOST          PID      SIZE  INTERFACES
slynux        pc1           4485     22852
slynux        pc2           4944     22852
slynux        pc3           5503     22852

Total virtual machines:     3    (you),       3    (all users).
Total consumed memory:    68556 KB (you),    68556 KB (all users).
slynux@gnubox:~$
```

Now we can set IP addresses for each of the machines using *ifconfig*:

```
pc1# ifconfig eth0 192.168.0.1
pc2# ifconfig eth0 192.168.0.2
pc3# ifconfig eth0 192.168.0.3
```

Try to ping between these machines—for example, from pc1 (with IP address *192.168.0.1*), *ping 192.168.0.2*.

A router is a network device used to bridge two networks of a different family of IP addresses. A network with a router to bridge two families of networks is shown in Figure 2.

We will now build a router using a virtual machine. First, start two virtual machines with network interfaces of different collision domains. Then set the IP addresses of the different families. Start a virtual machine with two interfaces of collision domains, and set IP addresses for the interfaces.

```
slynux@gnubox:~$ vstart pc1 --eth0=A // PC1
slynux@gnubox:~$ vstart pc1 --eth0=B  //PC2
slynux@gnubox:~$ vstart router --eth0=A --eth1=B    // Router
```

Execute the following commands in the virtual machines.

On R1:

```
router# ifconfig eth0 192.168.0.1
router# ifconfig eth1 192.168.1.1
```

On PC1:

```
pc1# ifconfig eth0 192.168.0.2
pc1# route add default gw 192.168.0.1  // Setting default gateway
```

On PC2:

```
pc1# ifconfig eth0 192.168.1.2
pc1# route add default gw 192.168.1.1 // Setting default gateway
```

Run the following ping tests to check if things are in place:

```
pc1# ping 192.168.0.1
pc1# ping 192.168.1.1
pc1# ping 192.168.1.2

pc2# ping 192.168.1.1
pc2# ping 192.168.0.1
pc2# ping 192.168.0.2
```

Now execute the *traceroute* command to find out the path of the network packet transfer.

On PC1:

```
pc1:~# traceroute 192.168.1.2
traceroute to 192.168.1.2 (192.168.1.2), 64 hops max, 40 byte packets
 1  192.168.0.1 (192.168.0.1)  4 ms  3 ms  6 ms
 2  192.168.1.5 (192.168.1.2)  1 ms  1 ms  1 ms
```

This means that it first passes through the gateway 192.168.0.1 and reaches the destination.

Figure 3 shows a network with two routers for advanced routing of different addresses.

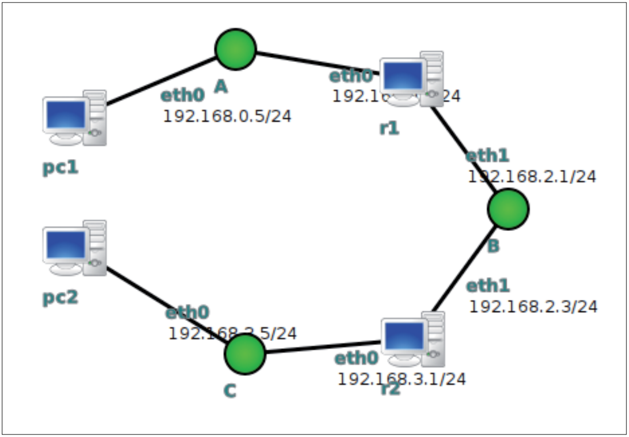

Figure 3: A network with two routers for advanced routing of different addresses

This experiment is more interesting, since we will use two routers and a bunch of *route* command executions to specify more address routes.

Let's start two VMs with interfaces belonging to collision domains A and C, and two routers R1 and R2 with collision domains of the interfaces A, B and B, C.

Execute the following commands:

```
slynux@gnubox:~$  vstart pc1 --eth0=A
slynux@gnubox:~$  vstart pc2 --eth0=C
slynux@gnubox:~$  vstart r1 --eth0=A --eth1=B
slynux@gnubox:~$  vstart r2 --eth0=B --eth1=C
```

On PC1:

```
pc1# ifconfig eth0 192.168.0.5
pc2# ifconfig eth0 192.168.3.5

r1# ifconfig eth0 192.168.0.1
r1# ifconfig eth1 192.168.2.1

r2# ifconfig eth1 192.168.2.3
r2# ifconfig eth0 192.168.3.1
```

Add the *routes*:

```
r1# route add -net 192.168.3.0 netmask 255.255.255.0 gw 192.168.2.1 dev
eth0
r2# route add -net 192.168.0.0 netmask 255.255.255.0 gw 192.168.2.3 dev
eth1

pc1# ping 192.168.3.5
pc2# ping 192.168.0.1
```

Let's now connect a network with the host machine gateway and access the Internet.

Each of the virtual machines used here are Debian Sid-based GNU/Linux installations. So we can update and install custom tools by providing connectivity to the virtual machines. We can share the Internet connection available on the host machine with the virtual machine using an interface argument.

Use the—*eth0=tap,TAP-ADDRESS,GUEST-ADDRESS* argument along with the *vstart* command to share the Internet connection. *TAP-ADDRESS* is the address attached with eth0 and *GUEST-ADDRESS* is the address attached for a new virtual interface created in the host machine.

The following example shows how to share an Internet connection with the virtual machine:

```
slynux@gnubox:~$ vstart pc1 --eth0=tap,192.168.1.94,192.168.1.98 --
eth1=A
```

Now you can *ping* from pc1 to your Internet gateway to check the connectivity.

Set *nameserver*, by executing the following command:

```
pc1# echo nameserver 192.168.0.1 > /etc/resolv.conf    //   192.168.0.1 is
the nameserver here.
pc1# apt-get update
```

You can now install additional applications to the virtual machine by using *apt-get*.

```
pc1# apt-get install  <package name>
```

Netkit Lab

Manually running *vstart* and a series of commands like *route*, *ifconfig*, etc, becomes very difficult and complex in the case of emulating comparatively large networks. It also requires doing the same thing again and again if we are to emulate the same network topology. Netkit Lab, an add-on script, is an exciting feature of the Netkit toolkit. You can write Netkit Lab configuration files such that by simply executing Netkit Lab, we can configure emulation on the network. Numerous virtual machines can be initialised from Netkit Lab configuration files.

Let us see how to make Netkit Lab work.

Create a directory (say, *netkit_testlab*) and include the following files in the *netkit_testlab* directory.

- *lab.conf* – this file consists of configuration details about the virtual machines like interfaces and collision domains.
- *<virtual_machine>.startup* – this file consists of commands to be executed initially while the virtual machine starts.
- *<virtual_machine>* – a blank directory with the same name as that of the virtual machine.

Let us set up a Netkit Lab to bridge two machines through a router with different IP address families:

```
slynux@gnubox:~$  mkdir netkit_testlab
slynux@gnubox:~$  cd netkit_testlab
slynux@gnubox:~/netkit_testlab$ mkdir pc1 pc2 r1
slynux@gnubox:~/netkit_testlab$ vim lab.conf
```

Type the following lines excluding the comments starting with // and save the *lab.conf* text file:

```
r1[0]="A"     // specifies  eth0, collision domain = A
r1[1]="B"   // specifies eth1, collision domain=B

pc1[0]="A"  // specifies eth0, collision domain=A
pc2[0]="B" //specifies eth0, collision domain=B
```

Type the commands to be executed during start up and save to a text file:

```
slynux@gnubox:~/netkit_testlab$  vim pc1.startup
ifconfig eth0 192.168.0.2
route add default gw 192.168.0.1

slynux@gnubox:~/netkit_testlab$  vim pc2.startup
ifconfig eth0 192.168.1.2
route add default gw 192.168.1.1

slynux@gnubox:~/netkit_testlab$  vim r1.startup
ifconfig eth0 192.168.0.1
ifconfig eth1 192.168.1.1
```

Now let's start the Netkit Lab using the *lstart* command:

```
slynux@gnubox:~$  cd netkit_testlab      // change current directory to
the lab directory
slynux@gnubox:~/netkit_testlab$  lstart
```

In order to shut down all the machines, use the *lhalt* command from the Netkit Lab directory:

```
slynux@gnubox:~/netkit_testlab$  lhalt
```

Try writing the lab *config* for more complex network topologies.

Netkit is arguably the best way to start experimenting with networking. Some universities are already using Netkit as a teaching aid. Through the UML-supported backbone, Netkit provides a real-time experience on emulation on network topologies. Find more bytes from *www.netkit.org*. Happy hacking till we meet again! END

By: Sarath Lakshman
The author is a Hacktivist of Free and Open Source Software from Kerala. He loves working on the GNU/Linux environment and contributes to the PiTiVi video editor project. He is also the developer of SLYNUX, a distro for newbies. He blogs at *www.sarathlakshman.info*