# Laboratory 1 Introduction to Netkit emulator

# Part1. Netkit Emulator

## What is Netkit emulator

Netkit is a network emulator based on **user-uode uinux (uml)**. Each emulated network device is a virtual linux box based on uml kernel. A user-mode linux process is also called **virtual machine (vm)**.

Each virtual machine has [1]:

- a console (a terminal window)
- a memory ("cut" into the memory of the host)
- a filesystem (stored in a single file of the host filesystem)
- (one or more) network interfaces

Each network interface can be connected to a (virtual) collision domain. Each virtual collision domain can be connected to several interfaces. Each virtual machine can be configured to play the role of a regular host, of a router, or a switch [1].

## Netkit Commands

Netkit provides users with two sets of commands [1]:

- ***v-prefixed commands*** (vcommands) – configure and start a single virtual machine
- ***l-prefixed commands*** (lcommands) – environment to set up a complex lab.

***Vcommands*** allow to startup virtual machines with arbitrary configurations:

- **vstart**: starts a new virtual machine
- **vlist**: lists 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**: "panic command" to clean up all netkit processes (including vms) and configuration settings on the host machine

***Lcommands*** allow to set up a complex lab consisting of several virtual machines:

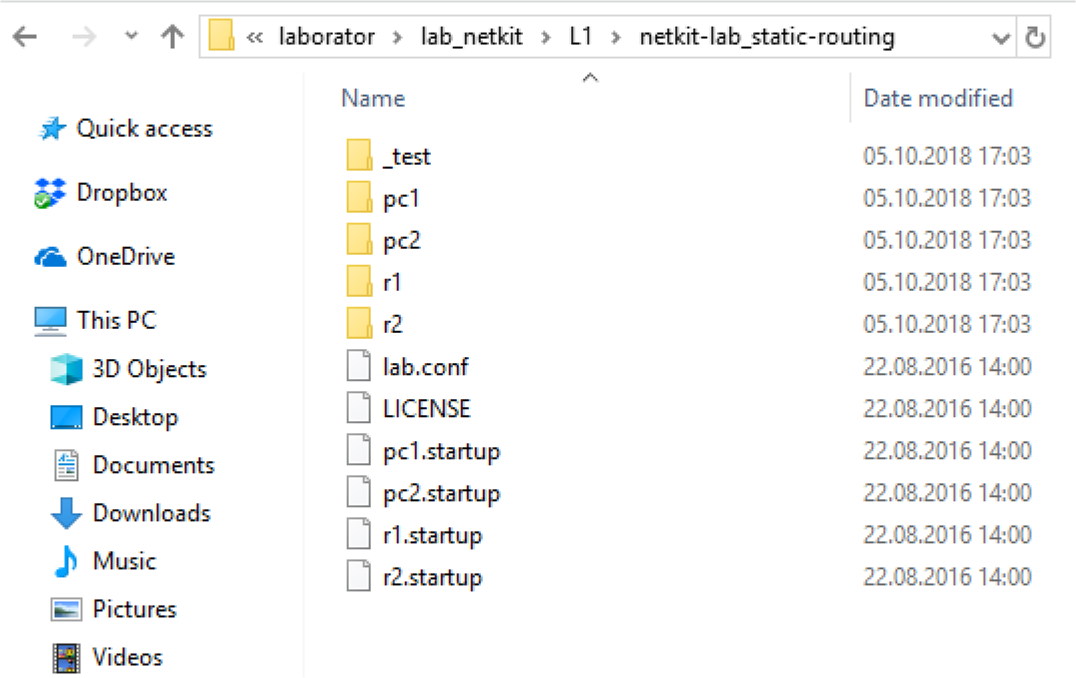
- **lstart**: starts a netkit lab
- **lhalt**: gracefully halts all vms of a lab
- **lcrash**: causes all the vms of a lab to crash
- **lclean**: removes temporary files from a lab directory
- **linfo**: provides information about a lab without starting it
- **ltest**: allows to run tests to check that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

## Preparing a Netkit lab

Emulating a network with Netkit is a matter of writing a simple file describing the link-level topology of the network to be emulated and some configuration files that are identical to those used by real world networking tools. Netkit then takes care of starting (emulated) network devices and of interconnecting them as required.

A netkit lab is a set of preconfigured virtual machines that can be started or halted together. A standard netkit lab is a directory tree containing (Figure 1):

- a `lab.conf` file describing the network topology
- a set of `subdirectories` that contain the configuration settings for each virtual machine
- `.startup` and `.shutdown` files that describe actions performed by virtual machines when they are started or halted
- [optionally] a `lab.dep` file describing dependency relationships on the startup order of virtual machines
- [optionally] a `_test directory` containing scripts for testing that the lab is working correctly



« laborator › lab_netkit › L1 › netkit-lab_static-routing

| Name | Date modified |
|---|---|
| _test | 05.10.2018 17:03 |
| pc1 | 05.10.2018 17:03 |
| pc2 | 05.10.2018 17:03 |
| r1 | 05.10.2018 17:03 |
| r2 | 05.10.2018 17:03 |
| lab.conf | 22.08.2016 14:00 |
| LICENSE | 22.08.2016 14:00 |
| pc1.startup | 22.08.2016 14:00 |
| pc2.startup | 22.08.2016 14:00 |
| r1.startup | 22.08.2016 14:00 |
| r2.startup | 22.08.2016 14:00 |

Figure 1: Netkit lab directory

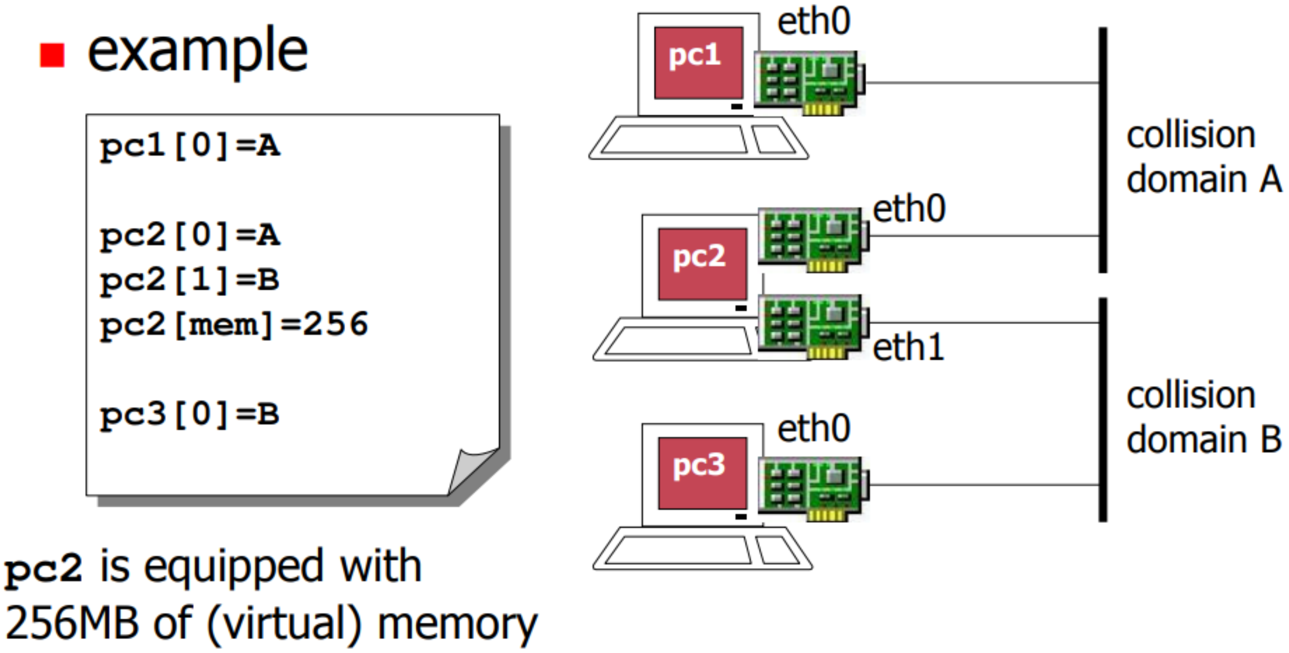
## Lab.conf

`Lab.conf` file is used to configure the network topology. It contrains [1]:

- the settings of the vms that make up a lab
- the topology of the network that interconnects the virtual machines of the lab

The file contains a list of assignments (Figure 2) with the format: `machine[arg] = value`

- `machine` is the name of the virtual machine (e.g., pc1)
- if `arg` is an integral number (say ***i***), then value is the name of the collision domain to which interface `eth`***i*** should be attached
- if `arg` is a string, then it must be the name of a `vstart` option and `value` is the argument (if any) to that option



example

```
pc1[0]=A

pc2[0]=A
pc2[1]=B
pc2[mem]=256

pc3[0]=B
```

pc2 is equipped with 256MB of (virtual) memory

Figure 2: lab.conf content for a given network topology [1]

Other optional assignments [1]:

- `machines="pc1 pc2 pc3…"`: explicitly declare the virtual machines that make up the lab
    - by default, the existence of a subdirectory `vm_name` in the labdirectory implies that a virtual machine `vm_name` is started
- LAB_DESCRIPTION
- LAB_VERSION
- LAB_AUTHOR
- LAB_EMAIL
- LAB_WEB

### Lab subdirectories

Netkit starts a virtual machine for each subdirectory, with the same name of the subdirectory itself

- unless `lab.conf` contains a `machines=` statement

The contents of subdirectory `vm` are mapped (=copied) into the root (/) of `vm's` filesystem

### Startup and Shutdown files

These files represent shell scripts that are used to tell virtual machines what to do when starting and shutting down. Startup scripts are used to configure the network interfaces or to start the network services.They are executed inside virtual machines.

- `shared.startup` and `shared.shutdown` affect all the virtual machines
- upon startup, a vm named `vm_name` runs

  - `shared.startup`
  - `vm_name.startup`
- upon shutdown, a vm named `vm_name` runs
  - `vm_name.shutdown`
  - `shared.shutdown`

### Lab.dep

A `lab.dep` file inside the lab directory describes dependencies and automatically enables parallel startup, e.g., "pc3 can only boot after pc2 and pc1 are up and running".

- `pc3: pc1 pc2`

### Launching/stopping a lab

The format for the command to launch or stop a lab is the following:

- `lcommand -d <lab_directory> [machine...]`

where lcommand can be one of the following:

- `lstart`, to start the lab
- `lhalt`, to gracefully shut down the virtual machines of a lab
- `lcrash`, to suddenly crash the virtual machines of a lab

## Part 2: Basic network configuration with netkit

Basic network topologies and network nodes configuration for a better understanding of the netkit environment will be performed in the experimental part of the lab.

## Simple network with two hosts interconnected

In this activity two host will be started and will be connected to the same collision domain. Their network interfaces will be configured in the same subnet: `192.168.1.0/24`. The connectivity between the machines will be tested with the `ping` command. The traffic exchange will be analysed with dedicated programs (`tcpdump` and `wireshark`).

### Network topology

The network topology is presented in Figure 3.

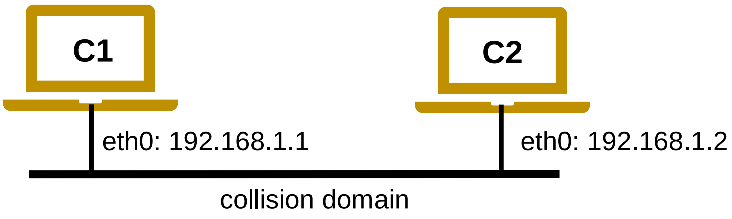


Figure 3: Network topology

## 1. Creating the virtual machines

The virtual machines will be started using `vstart` command:

- `vstart c1 -eth0=A`
- `vstart c2 -eth0=A`

The machines `c1` and `c2` will be started with the network interface `eth0` connected to the same `collision domain A`.

## 2. Configure the network interfaces

Use `ifconfig` command to check which interfaces are up on both machines.

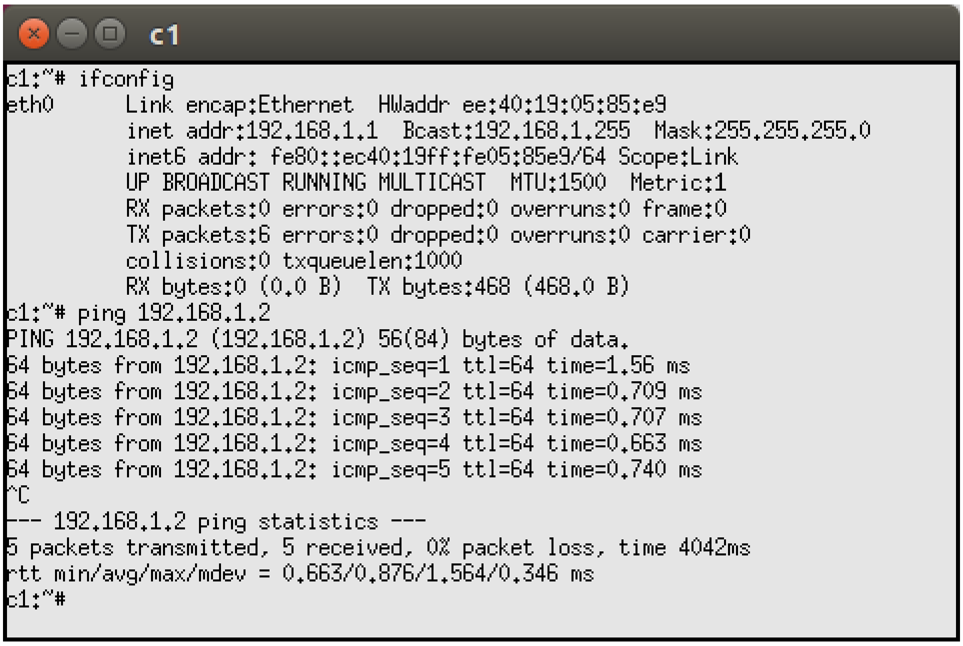
Configure the IP addresses given in Figure 1 on the `eht0` interfaces on `C1` and `C2` machines and bring them up using the ifconfig command:

- `ifconfig eth0` ***`ip_address_pc`*** `netmask` ***`netmask_pc`*** `up`

where ***`ip_address_pc`*** and ***`netmask_pc`*** represent the IP address and the netmask of the station (their values are given in Figure 3).

## 3. Test network connectivity with `ping` command

Use `ping` command to test network connectivity in both directions (Figure 4).



```
c1:~# ifconfig
eth0      Link encap:Ethernet  HWaddr ee:40:19:05:85:e9
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::ec40:19ff:fe05:85e9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:468 (468.0 B)
c1:~# ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=1.56 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.709 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.707 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.663 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=64 time=0.740 ms
^C
--- 192.168.1.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4042ms
rtt min/avg/max/mdev = 0.663/0.876/1.564/0.346 ms
c1:~#
```

Figure 4: Successfully ping the C2 machine

## 4. Analyse the packets using `tcpdump` command and `wireshark`

Tcpdump is a network sniffer that is used to capture and analyse the packets exchanged over the network. Use `man` command to see the documentation for `tcpdump` command:

- `c1:~# man tcpdump`

A fragment of the man command's output is shown in Table 1. It presents optional parameters that can be used with the command. With yellow are marked the ones that will be used:

- `[ -i interface ] –` specifies the interface to capture the packets
- `[ -s snaplen ] –` specifies the number of bytes captured per packet
- `[ -w file ] –` stores the pacjets to a file

```
TCPDUMP(8)                System Manager’s Manual                TCPDUMP(8)

NAME
       tcpdump - dump traffic on a network

SYNOPSIS
       tcpdump [ -AbdDefhHIJKlLnNOpqStuUvxX# ] [ -B buffer_size ]
               [ -c count ]
               [ -C file_size ] [ -G rotate_seconds ] [ -F file ]
               [ -i interface ] [ -j tstamp_type ] [ -m module ] [ -M secret ]
               [ --number ] [ -Q in|out|inout ]
               [ -r file ] [ -V file ] [ -s snaplen ] [ -T type ] [ -w file ]
               [ -W filecount ]
               [ -E spi@ipaddr algo:secret,…  ]
               [ -y datalinktype ] [ -z postrotate-command ] [ -Z user ]
               [ --time-stamp-precision=tstamp_precision ]
               [ --immediate-mode ] [ --version ]
               [ expression ]
```

Table 1: Tcpdump command’s parameters

Capture the packets on interface eth0 on virtual machines c2 while the ping command is used on virtual machine c1:

- `c1:~#ping 192.168.1.2`
- `c2:~#tcpdump -i eth0`

Q1: Analyze the pachets that are exchanged between the two virtual machines. What protocol is used? Which are the packets sent in each direction?

Capture the packets with `tcpdump` and store them in a local file `capture.pcap`. The file will be stored in the lab directory.

- `c2:~#tcpdump -i eth0 -w filelocation/filename`

filelocation/filename will be replaced with the appropriate path to lab directory and the filename.

Open the file `capture.pcap` with Wireshark packet analyzer. Analyze the content of the captured packets.

### 5. Generate the directory tree for the lab

Generate the same lab topology creating the associated directory tree. Generate the topology using the `lab.conf` file. Configure the IP addresses on the eth0 interfaces using the `.startup` files. Start the lab and check that the configurations are correct.

## References:

http://wiki.netkit.org/netkit-labs/netkit_introduction/netkit-introduction.pdf