Netkit: Easy Emulation of Complex Networks on Inexpensive Hardware

Maurizio Pizzonia
pizzonia@dia.uniroma3.it

Massimo Rimondini
rimondin@dia.uniroma3.it

Dept. of Computer Science and Automation
Roma Tre University

ABSTRACT

Network emulators are software environments that closely reproduce the functionalities and the behavior of real world networks.

In this paper we describe Netkit, a freely available lightweight network emulator based on User-Mode Linux. Netkit allows users to experiment with a large number of network technologies and provides tools for a straightforward setup of complex network scenarios that can be easily distributed via email or published on the Web. Netkit also comes with a set of ready to use experiences, accompanied by lecture slides, that enable users to immediately experiment with specific case studies. Our system has proved itself to be helpful in testing the configuration of ISP-scale real world networks and is profitably used within University level networking courses.

We provide a detailed comparison against other competing solutions and experimental measures about the scalability of the system.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.6 [**Computer-Communication Networks**]: Internetworking; K.3.0 [**Computers and Education**]: General

General Terms

Design, Experimentation

Keywords

Network emulation, Routing, Virtual laboratories, User-Mode Linux

1. INTRODUCTION

The emulation of networks is rapidly gaining the interest of network administrators, teachers and researchers in the networking area due to ease of setup, adherence to the behavior of real networks, and low cost. With a network emulation environment, administrators can quickly set up testbeds to check that particular configurations work as expected before deploying them on production networks, teachers can let students configure their own network for practice or examination by using their PC, and researchers can validate theoretical models with practical experimentation in environments that behave very much as the real ones.

Differently from a *simulator*, that allows the user to "compute" the evolution of a network, an *emulator* aims at closely reproducing the features and behavior of real world devices. It often consists of a software/hardware platform on which it is possible to run the same pieces of software that would be used on real devices. In an emulator, the network being tested undergoes the very same packet exchanges and state changes that would occur when using real devices. The drawback of this approach is that performance is limited by the efficiency of the software and of the underlying real hardware.

In this paper we describe Netkit [13, 25], a freely available network emulation environment based on User-Mode Linux [10, 36]. Netkit supports experimentation with a wide range of networking technologies, out of the box, and can be tweaked to support other technologies required for specific experiments. Netkit provides a collection of integrated, easy to use, and widely tested tools to simplify the setup of a virtual network. With respect to directly using a bare User-Mode Linux, Netkit tools make it very simple and straightforward to prepare laboratories (shortly *labs*) that implement complex network scenarios consisting of several emulated devices. A complete description of a lab consists of a set of plain ASCII files (usually no more than a few hundreds of kilobytes, highly compressible) that can be easily published over the Web or transferred by email. Netkit comes with a set of ready to use labs and teaching material that permit users to immediately experiment with specific case studies. Since Netkit strongly takes advantage of GNU/Linux, it supports most of the networking technologies available in this environment.

Netkit is also lightweight: for example, it is possible to launch a network experience consisting of 100 virtual machines in about 7 minutes on a typical workstation (Pentium 4 3.2GHz 2MB cache, 2GB RAM).

This paper is organized as follows. Section 2 illustrates the architecture of Netkit. Section 3 describes the user level tools and the procedure to set up labs. Section 4 shows a representative usage scenario of Netkit. Section 6 assesses

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
TRIDENTCOM 2008, 17th – 20th Mar 2008, Innsbruck, Austria.
Copyright © 2011 – 2012 ICST ISBN 978-963-9799-24-0
DOI 10.4108/icst.tridentcom.2008.3155

the scalability of Netkit by presenting a performance evaluation. Section 7 compares Netkit with other state of the art emulators. Conclusions are drawn in Section 8.

2. ARCHITECTURE AND SUPPORTED NETWORKING TECHNOLOGIES

Netkit is a lightweight network emulator based on open source software. It consists of several components: a kernel, a filesystem image, virtual hub software, and a set of user space commands. Netkit works out of the box: it includes everything that is needed to run an emulated network on a standard workstation and provides a set of ready to use virtual labs that can be used to experiment with interesting case studies. Netkit is conceived for easy installation and usage and does not require administrative privileges for either one of these operations.

Netkit emulated devices are based on the User-Mode Linux (UML) kernel [10, 36], a port of the standard Linux kernel designed to run as a user space process on the real machine (*host*). An instance of UML provides a *virtual machine*, namely an environment having its own processes that perform I/O by interacting with the UML kernel instead of the host kernel. Regardless of the hardware configuration of the host, a virtual machine can be equipped with arbitrarily chosen devices, including disks and network interfaces. A virtual machine can then play the role of a specific device (e.g., a router) by running appropriate software (e.g., Quagga [35], XORP [8]).

Starting the UML kernel involves dealing with long and complex command lines. For this reason, Netkit provides a set of tools that allow users to easily configure and set up complex network labs consisting of several virtual devices. Section 3 describes these tools in detail.

Each virtual machine has its own filesystem which contains a full-fledged GNU/Linux installation, based on the Debian [29] distribution and suitably tuned to operate inside UML and to interface with Netkit's commands. The filesystem is stored inside a *backing* file on the host, whose size is approximately 600MB. However, in Netkit it is possible to run a complex network scenario without having to use one backing file for each virtual machine. In Netkit, each virtual machine reads from the same backing file but writes its changes to its own *COW* file following a *Copy On Write* approach. The UML kernel takes care to show a consistent view of the filesystem inside each virtual machine. The size of a COW file is typically around 10MB. Therefore, a Netkit installation takes about 600MB of disk space for the backing file plus about 10MB for each started virtual machine. The requirements in terms of main memory are also small: each running virtual machine needs about 15MB of memory in the default configuration.

Even if Netkit only supports experimentation with GNU/Linux virtual machines, this environment offers a very wide spectrum of networking technologies.

Virtual machines can be interconnected by using *virtual hubs* [14, 37], namely software running on the host that emulates ethernet collision domains. Optionally, a virtual hub can be configured to access an external network, e.g., to connect virtual machines to the Internet. Fig. 1 shows an example in which virtual machines `vm1` and `vm2` are connected to virtual hub `A`. Virtual machine `vm2` has two virtual network interfaces and is also connected to virtual hub `B` along with

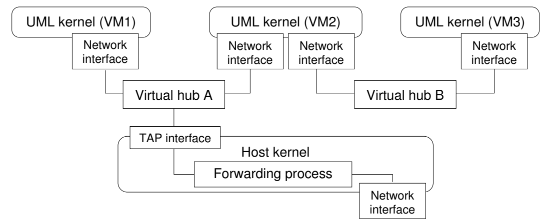

Figure 1: How Netkit virtual machines are networked, possibly with a connection to an external network.

```
Host console
vstart vm1 --eth0=tap,10.0.0.1,10.0.0.2
vstart vm2 --eth0=tap,10.0.0.1,10.0.0.3 --eth1=B
vstart vm3 --eth0=B
```

Figure 2: Netkit commands to implement the topology in Fig. 1. The special name "tap" for collision domain A indicates that the corresponding virtual hub should be connected to the TAP interface.

`vm3`. By running appropriate software, `vm2` can permit `vm1` to communicate with `vm3`. In Netkit `vm2` can be configured to operate as a switch, as a router, as a firewall, as a Web proxy, etc. In this setting, `vm1` and `vm2` can also reach an external network by means of virtual hub `A` that is connected to the special TAP interface on the host. A network tap is a device that provides a way to access the data flowing over a network link. In Netkit, TAP is a device driver that makes it possible to attach a virtual network interface to a userspace process (in this case, the virtual hub). The host should take care of routing packets between the TAP interface and the real network interface. The Netkit commands in Fig. 2 automatically set up the configuration of Fig. 1, including NAT translation rules so as to allow any IP address to be used in the emulated network. The addresses used by `vm1` and `vm2` to access the Internet are also automatically configured.

Although the physical layer in Netkit is limited to ethernet emulation, a large number of other networking technologies are supported as in a regular Linux machine. Among the technologies supported by the UML kernel currently shipped with Netkit there are 802.1d bridging and spanning tree, 802.1Q VLAN tagging, IPv4, IPv6, and MPLS based forwarding, ARP, ICMP, UDP, TCP, IP filtering and mangling (e.g., NAT), IPsec (transport and tunnel mode, ESP and AH), GRE tunnels, load balancing by equal cost multipath, and multicast with PIM-SM. These technologies can be configured and managed using the traditional utilities available under GNU/Linux. Kernel level support for other technologies can be obtained by expert users by building a custom UML kernel. Netkit supports this activity by providing everything that is needed to re-build the shipped kernel from scratch. Selection of the kernel for each virtual machine is supported when multiple kernels are available.

A wide range of technologies are implemented by software installed in the filesystem shipped with Netkit. Among them there are DHCP, PPP, DNS server (bind), HTTP and HTTPS (apache), Web proxy (squid), email (exim), FTP, NFS, Samba, Telnet, SSH. Among the supported routing protocols there are RIP, OSPF, IS-IS, BGP (Quagga, XORP), providing MIBs accessible via SNMP. Among the

supported security related technologies there are RADIUS, PAM, IKE (openswan and racoon), and the Snort network intrusion detection system. Also, traffic can be forged, captured, and analyzed by means of tcpdump, tethereal, ssldump, tcpreplay, tcpreen, sendip, hping, dsniff, and ettercap. Within each virtual machine scripting languages are also available including bash, expect, awk, and perl. For a complete list of installed packages, see [33]. If specific experiments require software that is not available in the shipped filesystem, it is possible to quickly grab and install new packages from the Internet by using the Debian package management system `apt` [3].

Netkit is distributed in the form of three packages. The core of Netkit is a collection of shell scripts that manage instances of the UML kernel and of the virtual hub software transparently with respect to the end user. The choice of using shell scripts results in easier maintenance for the developers and the possibility for advanced users to quickly apply changes to suit the needs of a specific experiment. Care has been taken in ensuring compatibility with a wide range of Linux distributions: Netkit scripts only rely on tools that are available in most basic Linux installations, and are tweaked to be compatible with POSIX compliant system shells (e.g., dash). Several error checks are spread in the scripts to provide end users with meaningful error messages in case something goes wrong. The core package also contains man pages for the scripts and a statically compiled release of the virtual hub software.

The other packages provided with Netkit are the filesystem and the kernel. The filesystem package contains a disk image with a single ext2 partition containing an extensively tested Debian installation. Empty areas of the filesystem have been wiped before building the package to support better compression. The kernel package consists of a vanilla kernel [32] compiled to run in user mode and some patches that have been applied to better integrate the kernel with Netkit. Kernel modules are also provided, and are intentionally kept out of the filesystem image in order to allow developers to update the kernel and the filesystem separately. The kernel has been carefully configured so that users can enable specific features by simply loading modules, and comes with default settings that support most basic experiments. With these settings, virtual machines generate timer interrupts at a rate that makes emulated time match as closely as possible the wall clock. Therefore, to a certain extent, packet timings reflect real world timings. However, depending on the policy adopted by the scheduler on the host machine, misalignments may still occur. The configuration file used during the kernel compilation is also provided in the kernel package to support the creation of customized kernels to be used in specific experiments.

The reason why Netkit consists of three separate packages is that, in this way, users do not need to download a large filesystem image every time an update is released: instead, they can selectively get only the component that has undergone changes.

Further details about the architecture of Netkit are provided in [13].

3. TOOLS FOR SETTING UP EMULATED NETWORKS

The user interface of Netkit consists of a set of commands.

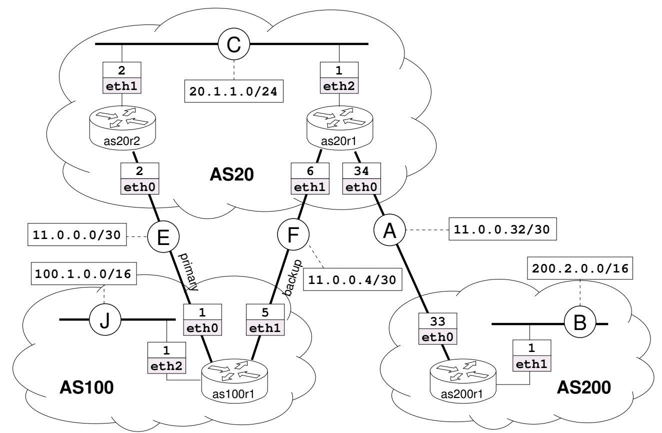

Figure 3: A sample network that can be implemented as a Netkit lab. Each collision domain is labeled with a capital letter and its subnet. Each router interface is labeled with the interface name and the last byte of its IP address.

Some of them, whose name starts with a `v` (*vtools* in the following), can be used to configure single virtual machines, while others, whose name starts with an `l` (*ltools*), support easy setup of virtual labs implementing complex scenarios. All the commands are fully documented by man pages that are installed with Netkit.

Command `vstart` allows users to configure and start a virtual machine identified by an arbitrary name. Customizable parameters include the amount of available memory, the kernel and filesystem to be used, the number of network interfaces and the collision domains they are attached to. Netkit's default settings usually fit most of the needs, so that starting a virtual network device often simply consists in specifying the network interfaces it should be equipped with. `vstart` takes care of starting UML kernel instances as well as the required virtual hubs. If requested, `vstart` can also apply the configuration needed to make a virtual machine access an external network. Once a virtual machine has started up, it can be configured for networking by using regular Linux commands (e.g., `ifconfig` for IP addresses, `ip` or `route` for static routes, `brctl` for bridging, etc.).

Other Netkit commands allow users to get information about currently running virtual machines (`vlist`) and to stop them (`vcrash` and `vhalt`).

The vtools can be profitably used for configuring, starting, and managing few virtual machines, but the setup of a complex experience usually involves many more configurations than just virtual machine settings. The ltools provide a user interface to easily set up, manage, and shut down a virtual lab consisting of several virtual machines in a straightforward way. A Netkit *lab* is a set of fully preconfigured virtual machines that can be started (`lstart` command) and stopped (`lcrash` and `lhalt` commands) as a whole.

A lab is described by a collection of files and directories on the host. The presence of a top-level directory in a lab instructs Netkit to start a virtual machine named as the directory itself. For example, the network in Fig. 3, consisting of machines `as100r1`, `as200r1`, `as20r1`, and `as20r2`, is implemented by the directory structure shown in Fig. 4.

Files and directories under each top-level directory are

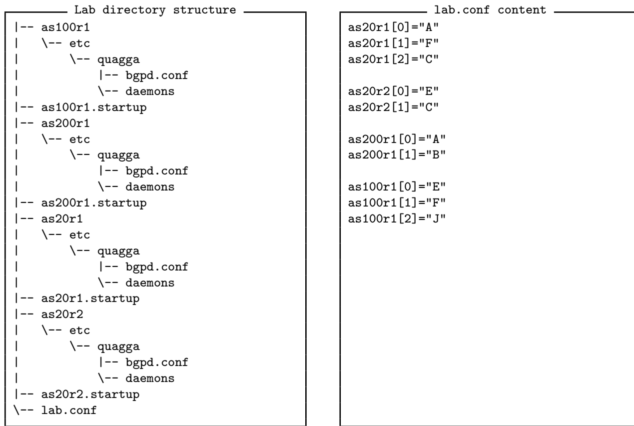

Figure 4: Directory structure for the lab shown in Fig. 3 and content of the file `lab.conf` describing the physical topology.

automatically copied to the root (`/`) of the filesystem of the corresponding virtual machine upon its startup. In this way, configuration files needed for specific services can be simply placed on the host and they are automatically made available inside the virtual machines. For example, in the lab in Fig. 4 the directory `/etc/quagga` inside each virtual machine will be automatically populated with the Quagga configuration files specified in the lab.

A file `lab.conf` describes the link level topology as well as other configuration parameters for the virtual machines. Fig. 4 shows a sample `lab.conf` for the topology in Fig. 3. A line `as20r1[0]=A` indicates that virtual machine `as20r1` will be equipped with a network interface `eth0` attached to collision domain `A`.

Virtual machine `vm` automatically executes files `vm.startup` and `vm.shutdown` on its startup and shutdown phase, respectively. These two files can be used to automatically apply settings (e.g., configure IP addresses) or start services. Also, dependencies on the startup order of virtual machines can be described in a file `lab.dep`.

To support the development of new releases of the labs, and of Netkit itself, the labs can be equipped with user defined self testing procedures to dump significant information about the status of virtual machines. The `ltest` command can be used to easily save the dump as a signature of a correctly running emulated network. `ltest`, along with the standard `diff` utility, can be used to easily perform regression tests.

Netkit comes with a set of ready to use labs [25] implementing representative network scenarios. The labs cover basic topics such as the ARP and RIP protocols, advanced topics such as bridging and spanning tree computation, application level services including DNS and email, and several scenarios of interdomain routing with BGP. Each lab is supported by a set of lecture slides that introduce the topic and suggest insightful experiments that can be performed on the lab itself.

Fig. 5 shows a working session with the “Small Internet” lab available at [25], consisting of 14 virtual machines. This lab takes about 2 minutes to start and less than 20 seconds to stop on a standard workstation (Pentium 4 3.2GHz 2MB cache, 2GB RAM). The window with black background in the upper right corner in Fig. 5 is a terminal window on the

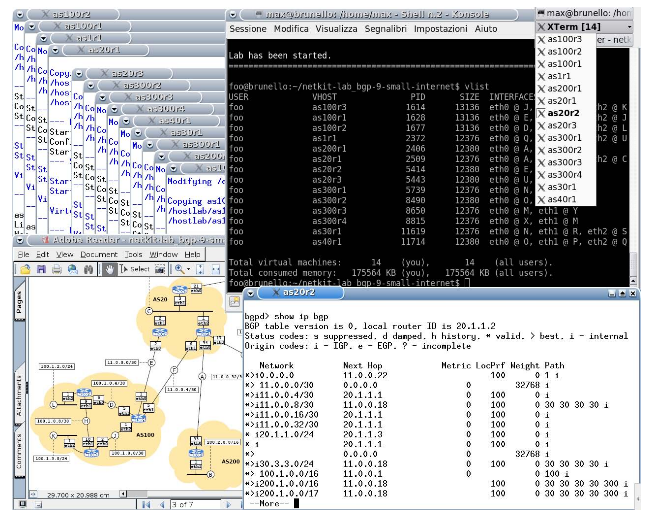

Figure 5: A typical working session with Netkit.

host. It contains a list of currently running virtual machines generated by `vlist`. Each virtual machine has its own terminal window which title is the name of the virtual machine itself. Switching to a virtual machine is as simple as clicking on its window. The window in the lower right shows the routing software Quagga [35] running on virtual machine `as20r2`: the `show ip bgp` command is used to display the routing table of the BGP protocol on that router.

4. CASE STUDY: BGP MULTIHOMING

Several networking scenarios implemented using Netkit are available on the Netkit website [25] in the form of teaching material. In this section we analyze a representative Netkit scenario that can be used to study routing protocols. The lab described here shows how to configure a multihomed customer with a primary link and a backup link to an upstream provider. The multihoming is implemented using the BGP protocol [11,40], which is the de facto standard for interdomain routing. The lab also points out issues in the interaction between intradomain and interdomain routing that are difficult to reproduce without real routing software. In particular, BGP only selects a route as best if a recursive lookup on the next-hop succeeds, i.e., the path to reach the next hop is known by static routes or some IGP protocol, as stated in [39]. We show how to use Netkit in order to examine the impact of failed recursive lookups.

The topology of the lab is shown in Fig. 3. AS100 is a customer AS having two links to its provider AS20. One of the links is used as a backup: traffic flows through it only if the other link fails. The routers in AS20 exchange routes via an iBGP peering. AS100 is configured in order to avoid providing transit across its links to the provider. Instead, AS20 provides transit between AS100 and its single homed customer AS200.

Experimenting with such a configuration is easy inside an emulator that runs on a regular PC, while 4 routers with BGP support are usually not available for experimentation, at least in many didactic and working environments. A simulator would not fit well the objectives of this lab, as the tricky interplay between intradomain and interdomain routing protocols is usually not fully modeled in such systems.

```
as100r1's bgpd.conf
1.  router bgp 100
2.  network 100.1.0.0/16
3.  neighbor 11.0.0.2 remote-as 20
4.  neighbor 11.0.0.2 description Router as20r2 (primary)
5.  neighbor 11.0.0.2 prefix-list defaultIn in
6.  neighbor 11.0.0.2 prefix-list mineOutOnly out
7.  neighbor 11.0.0.6 remote-as 20
8.  neighbor 11.0.0.6 description Router as20r1 (backup)
9.  neighbor 11.0.0.6 prefix-list defaultIn in
10. neighbor 11.0.0.6 prefix-list mineOutOnly out
11. neighbor 11.0.0.6 route-map localPrefIn in
12. neighbor 11.0.0.6 route-map metricOut out
13. !
14. access-list myAggregate permit 100.1.0.0/16
15. !
16. ip prefix-list defaultIn seq 5 permit 0.0.0.0/0
17. ip prefix-list mineOutOnly seq 5 permit 100.1.0.0/16
18. !
19. route-map metricOut permit 10
20.  match ip address myAggregate
21.  set metric 10
22. !
23. route-map localPrefIn permit 10
24. set local-preference 90
```

Figure 6: A fragment of the configuration of BGP on as100r1.

```
as100r1
as100r1:~# telnet localhost bgpd
Trying 127.0.0.1...
Connected to as100r1.
Escape character is '^]'.

Hello, this is zebra (version 0.94).
Copyright 1996-2002 Kunihiro Ishiguro.


User Access Verification

Password:
bgpd> show ip bgp
BGP table version is 0, local router ID is 100.1.0.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 0.0.0.0          11.0.0.2                               0 20 i
*                   11.0.0.6                      90       0 20 i
*> 100.1.0.0/16     0.0.0.0                  0         32768 i

Total number of prefixes 2
```

Figure 7: Interaction with the terminal window of as100r1. The figure shows the BGP routing table known by Quagga.

Fig. 4 shows the files and directories that make up the lab. It can be easily seen from the names of the configuration files that routers in this lab run the BGP routing protocol. The physical topology of Fig. 3 is implemented in the file `lab.conf` as shown in Fig. 4.

Fig. 6 shows a portion of the `bgpd.conf` file of router `as100r1`. The backup policy is enforced by `as100r1` by using local-preference (lines 11, 23, and 24) and Multi-Exit-Discriminator (lines 12, 14, and 19–21). Lines 5-6, 9-10, and 16-17 prevent traffic of the provider from traversing the customer AS100.

After starting the lab with `lstart`, a virtual machine can be selected by simply clicking on its window. Fig. 7 shows how the user would interact with `as100r1`. The `telnet` command is used to contact the Quagga BGP daemon. After entering the password, Quagga offers a prompt which accepts commands that are similar to those of real world routers. For example, in Fig. 7 the **`show ip bgp`** command is used to display the BGP routing table of `as100r1`. It is easy to notice that `as100r1` is offered two instances of the default route and chooses the one through the primary link E and next-hop `11.0.0.2` to actually forward traffic (indicated by '`>`' in Fig. 7).

The BGP process on `as20r2` is unable to select any route to `200.2.0.0/16`. This is shown in Fig. 8 (lower window) by the fact that the only route to `200.2.0.0/16` is not marked

```
as20r2
Router> show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       B - BGP, > - selected route, * - FIB route

C>* 11.0.0.0/30 is directly connected, eth0
B>* 11.0.0.4/30 [200/0] via 20.1.1.1, eth1, 00:00:04
B>* 11.0.0.32/30 [200/0] via 20.1.1.1, eth1, 00:00:04
C>* 20.1.1.0/24 is directly connected, eth1
B>* 100.1.0.0/16 [20/0] via 11.0.0.1, eth0, 00:00:06
C>* 127.0.0.0/8 is directly connected, lo
Router>
```

```
as20r2
   Network          Next Hop            Metric LocPrf Weight Path
* i0.0.0.0          20.1.1.1                 0    100      0 i
*>                  0.0.0.0                  0         32768 i
*> 11.0.0.0/30      0.0.0.0                  0         32768 i
*>i11.0.0.4/30      20.1.1.1                 0    100      0 i
*>i11.0.0.32/30     20.1.1.1                 0    100      0 i
* i20.1.1.0/24      20.1.1.1                 0    100      0 i
*>                  0.0.0.0                  0         32768 i
* i100.1.0.0/16     11.0.0.5                10    100      0 100 i
*>                  11.0.0.1                 0             0 100 i
* i200.2.0.0/16     11.0.0.33                0    100      0 200 i

Total number of prefixes 7
bgpd>
```

Figure 8: Forwarding table and BGP routing table on as20r2.

```
as20r1.startup
/sbin/ifconfig eth0 11.0.0.34 netmask 255.255.255.252 broadcast 11.0.0.35 up
/sbin/ifconfig eth1 11.0.0.6 netmask 255.255.255.252  broadcast 11.0.0.7 up
/sbin/ifconfig eth2 20.1.1.1 netmask 255.255.255.0 broadcast 20.1.1.255 up
route add -net 11.0.0.0/30 gw 20.1.1.2 dev eth2
/etc/init.d/quagga start
```

```
as20r2.startup
/sbin/ifconfig eth0 11.0.0.2 netmask 255.255.255.252 broadcast 11.0.0.3 up
/sbin/ifconfig eth1 20.1.1.2 netmask 255.255.255.0 broadcast 20.1.1.255 up
route add -net 11.0.0.32/30 gw 20.1.1.1 dev eth1
route add -net 11.0.0.4/30 gw 20.1.1.1 dev eth1
/etc/init.d/quagga start
```

Figure 9: Commands to instruct as20r1 and as20r2 to configure static routes upon startup.

by a '`>`', even if this route has actually been learned by an iBGP session. The reason of this behavior is that the next-hop `11.0.0.33` is only reachable via BGP, as shown in the upper window by the '`B`' flag. Therefore, `as20r2` is unable to reach `as200r1`. A similar problem occurs on `as20r1`, which is unable to reach `11.0.0.1` via any IGP. Also consider that, in case of failure of link E, `as20r2` fails to recursively look up `11.0.0.5` and loses connectivity to `100.1.0.0/16`.

Several configurations on the routers of AS20 can solve this problem: (i) adding static routes, (ii) enabling an IGP (e.g., RIP, OSPF), and (iii) using the BGP **`next-hop-self`** directive. Netkit supports all these approaches. In any case, the new configurations can be applied on the fly by directly interacting with the terminals of the virtual machines or can be implemented in the lab, in which case they take effect when the lab is next restarted. If required, Netkit also allows users to selectively halt and restart only the virtual machines that are affected by the changes.

We choose to modify the lab adding static routes. The lab files `as20r1.startup` and `as20r2.startup` are changed in order to configure the routes on startup. In Fig. 9 the lines added to the files have been highlighted. Once the changes have been made, `as20r1` and `as20r2` can be restarted by using the commands `lcrash as20r1 as20r2` and `lstart as20r1 as20r2`.

Fig. 10 shows that the route to `200.2.0.0/16` is now selected by BGP on `as20r2`. This is indicated by '`>`' in the BGP routing table (lower window). Since the reachability of `11.0.0.32/30` has been set up by the `route` commands

```
as20r2
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       B - BGP, > - selected route, * - FIB route

C>* 11.0.0.0/30 is directly connected, eth0
B>* 11.0.0.4/30 [200/0] via 20.1.1.1, eth1, 00:00:13
B   11.0.0.32/30 [200/0] via 20.1.1.1, eth1, 00:00:13
K>* 11.0.0.32/30 via 20.1.1.1, eth1
C>* 20.1.1.0/24 is directly connected, eth1
B>* 100.1.0.0/16 [20/0] via 11.0.0.1, eth0, 00:00:13
C>* 127.0.0.0/8 is directly connected, lo
B>* 200.2.0.0/16 [200/0] via 11.0.0.33, recursive via 20.1.1.1, eth1, 00:00:08
```

```
as20r2
   Network          Next Hop            Metric LocPrf Weight Path
* i0.0.0.0          20.1.1.1                 0    100      0 i
*>                  0.0.0.0                  0         32768 i
*> 11.0.0.0/30      0.0.0.0                  0         32768 i
*>i11.0.0.4/30      20.1.1.1                 0    100      0 i
*>i11.0.0.32/30     20.1.1.1                 0    100      0 i
* i20.1.1.0/24      20.1.1.1                 0    100      0 i
*>                  0.0.0.0                  0         32768 i
*> 100.1.0.0/16     11.0.0.1                 0             0 100 i
*>i200.2.0.0/16     11.0.0.33                0    100      0 200 i

Total number of prefixes 7
```

Figure 10: Forwarding table and BGP routing table on as20r2 after adding static routes to permit recursive lookup.

```
as20r2
bgpd> enable
Password:
bgpd# configure terminal
bgpd(config)# router bgp 20
bgpd(config-router)# neighbor 11.0.0.1 shutdown
bgpd(config-router)# exit
bgpd(config)# exit
bgpd# exit
```

Figure 11: How to administratively shut down a BGP peering in Netkit.

```
as20r2
* i20.1.1.0/24      20.1.1.1                 0    100      0 i
*>                  0.0.0.0                  0         32768 i
*>i100.1.0.0/16     11.0.0.5                10    100      0 100 i
*>i200.2.0.0/16     11.0.0.33                0    100      0 200 i

Total number of prefixes 7
```

Figure 12: The BGP routing table of as20r2 after the failure of the primary link E.

highlighted in Fig. 9, the selected entry for `11.0.0.32/30` in the forwarding table is now correctly recognized as statically configured, and is therefore marked with the flag '`K`' (Fig. 10, upper window).

Observe that, in turn, `as20r1` is now able to perform a successful lookup on the next-hop `11.0.0.1`. Therefore, `as20r1` can now correctly choose to reach `100.1.0.0/16` through link E and propagates only this choice to its iBGP neighbor `as20r2`.

To experiment with the backup configuration, we force traffic away from link E. This can be achieved in Netkit by shutting down one of the network interfaces at its endpoints using `ifconfig` or by administratively shutting down the BGP peering on that link. Fig. 11 shows the BGP commands that implement the latter solution by bringing down the peering on link E.

After waiting for a few seconds, `as20r2` updates its BGP routing table and starts using link F as shown in Fig. 12. Bringing link E up again restores the original path.

5. APPLICATIONS AND USAGE SCENARIOS

In Section 4 we have described just one use-case scenario of Netkit. Many other scenarios can be imagined, both in the research context and in the instructional one. For example, the BGP protocol is well known to be subject to routing oscillations [21]. This is usually caused by unforeseen interactions among routing policies deployed at different Autonomous Systems. Moreover, bad interactions between BGP and an interior gateway protocol can lead to forwarding loops [22]. A researcher or an operator could take advantage of Netkit to study interdomain routing protocol issues, including routing oscillations and forwarding loops. While theoretical models would provide a static description of the conditions that trigger oscillations, Netkit could be used as a support tool to validate and extend the theoretical models based on the observation of routing dynamics.

Researchers interested in developing routing protocols can use Netkit as a platform to implement and debug their prototypes without jamming a real network. The same objective would be much more difficult to achieve on a real device, at the very least because the source code of proprietary firmware is usually not available.

Other possible usage scenarios for Netkit, probably more targeted at operators, include the definition of an address plan or the deployment of security countermeasures. For example, the setup of multiple levels of firewall or NAT would be simplified by having the possibility to observe in a Netkit lab the combined effect of specific configurations.

As an example of real world scale emulation, Netkit has been successfully used to emulate the Italian Academic & Research Network (Consortium GARR [31]).

One of the most interesting contexts of application of Netkit is definitely didactics. Netkit offers students the otherwise unfeasible opportunity to exercise networking concepts on a live network on their own, with the possibility of freely accessing a range of technologies but without the need to fiddle with real devices. The effectiveness of using Netkit for instruction has been proved over the years. In fact, at present Netkit is being profitably used in several editions of networking courses at the Calabria University and at the Pisa University, within training courses for teachers organized by the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR), and within Linux@School, a project supported by the MIUR and the Polytechnic of Milan to spread the usage of free software in the schools. Netkit is being pervasively used within networking courses at the Roma Tre University, where students can learn basic and advanced networking concepts with the help of well documented virtual labs. Moreover, at Roma Tre Netkit is also used as a platform to give exams: in this way students can prove that they actually learned how to setup a network.

6. PERFORMANCE AND SCALABILITY

In this section we present an evaluation of the performance of Netkit. In particular, we consider emulated network scenarios consisting of an increasing number of nodes and measure the startup time of the emulation. We also estimate resource consumption during an experiment of file transfer over a virtual network.

We performed the tests using Netkit version 2.5, kernel version K2.3 (2.6.23.1), and filesystem version F2.2. For each test we generated a new lab with a fixed number of virtual machines. Each virtual machine was equipped with a single network interface and ran the Quagga and RIP routing daemons with a default configuration. All the network interfaces were connected to the same collision domain. Each virtual machine was configured to have 20MB of avail-

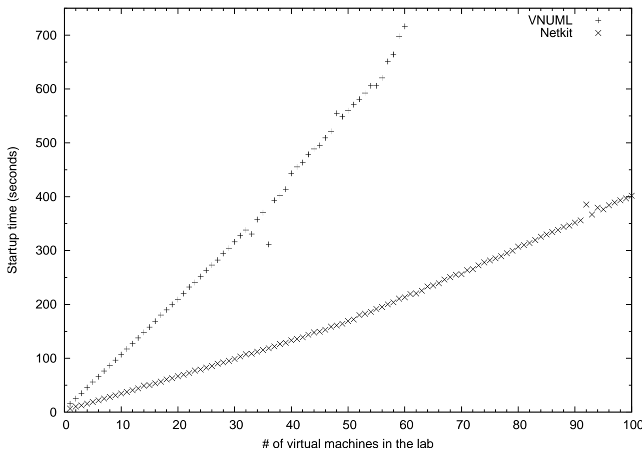

Figure 13: Startup time of an emulated network of varying size.

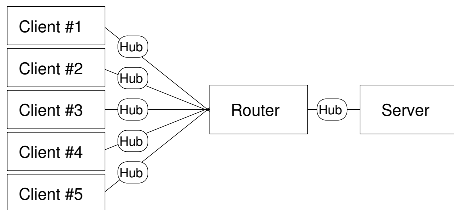

Figure 14: Topology of the virtual network used to test CPU load.

able memory, causing UML to consume up to 24MB on the host (actual usage depends on the processes running in the virtual machine). The lab was then started using the Netkit scripts and its startup time evaluated. The disk cache was completely cleared before starting each test. To prevent the overhead of graphical interfaces from influencing the timings, Netkit virtual machines were started without a terminal window. The tests have been performed on a Pentium 4 3.2GHz workstation with 2MB of cache for the CPU, 2GB of RAM, and no swap space configured.

Fig. 13 shows the results of the tests. It can be easily seen that the time required by each lab to start up grows linearly with the number of virtual machines. This means that every virtual machine takes the same amount of time to boot, regardless of the number of already running virtual machines. Therefore, idle virtual machines do not consume CPU resources on the host and the startup time only changes because the number of virtual machines that have to be booted increases for each lab.

We performed the same kind of test on the same machine using the latest stable release of VNUML (1.8.1), with root filesystem version 0.5.1. This release runs a 2.6.18.1 UML kernel. One may argue that the tests should have been performed using the same kernel and filesystem for both Netkit and VNUML. However, our goal was to compare the two products as they are, using the configuration and support tools that come with their own packages.

For each test with VNUML, we generated an XML specification of a network with exactly the same topology and configuration as for the case of Netkit. Virtual machines were configured with the default amount of memory: about

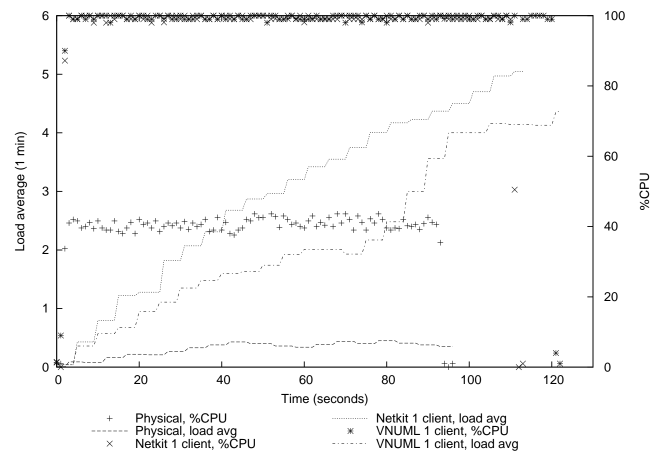

Figure 15: CPU load during a file transfer from a server to a single client.

27MB available inside the virtual machine, corresponding to a maximum of 32MB consumed on the host. The emulation was then started using the VNUML parser and the overall startup time measured. We chose to limit the size of the labs used in the test to 60 virtual machines, as this was enough to compare VNUML with Netkit.

By looking at Fig. 13, it is clear that the startup time of a VNUML emulation is higher than that of Netkit. This is mostly due to the number of services that are started in the virtual machines at boot time. This difference can be further appreciated by noticing that a single Netkit virtual machine runs by default about 20 processes on the host, while a VNUML virtual machine runs about 30 processes.

However, in order to better assess the resource consumption during an emulation, we also measured the CPU load on the host during a large file transfer via HTTP. We first set up two phisically different hosts connected by a 100Mbps link, one running an Apache server and one running a text mode browser. We then downloaded a 1GB file and measured the CPU usage on the client during the file transfer. To increase the accuracy of the measurement, the server was forced to fully cache the file before starting the transfer, and the data downloaded by the client was not saved to any file but simply discarded (we used `/dev/null` as target). The same file transfer was then repeated on a virtual network consisting of a server node, an intermediate router, and a set of client nodes, as depicted in Fig. 14. The virtual setting was implemented first with a single client, and then with all the 5 clients downloading the same file simultaneously. We implemented the scenario both in Netkit and in VNUML. All the experiments were run on a single CPU Pentium 4 3.2GHz workstation with 2MB of CPU cache and 2GB of RAM.

Figures 15 and 16 show the results we obtained. We sampled, at a 1 second rate, the percentage of CPU usage (including system time) and the average load (number of processes in running or ready state) over a 1 minute interval. Fig. 15 shows that the time required by the file transfer to complete in the case of the physical network and of virtual networks with a single client is more or less the same: in fact, the curves disappear after about 2 minutes. The percentage of CPU used by Netkit and used by VNUML is comparable and always around 100%. In the case with 5 clients, all Net-

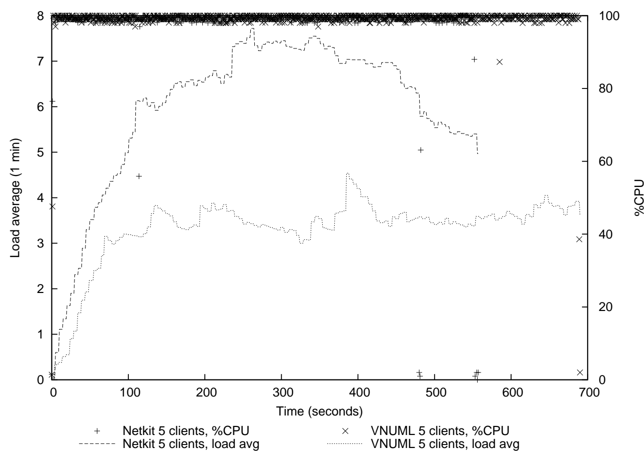

Figure 16: CPU load during simultaneous file transfers from a server to 5 clients.

kit virtual machines complete the transfer in less than 10 minutes, while in VNUML this takes more than 11 minutes. While the CPU usage is again comparable, VNUML exhibits lower load averages. Therefore, VNUML processes are likely to spend more time waiting for input/output operations.

According to our results, we can conclude that Netkit provides better scalability than VNUML, and enables to run emulated scenarios consisting of even hundreds of nodes with a yet reasonable load on the host.

Further performance evaluations presented in [28] confirm that a Netkit emulation has a limited resource usage compared with other products.

7. RELATED WORK

There are many network emulation solutions available, which are based on different emulation technologies. This section provides only a brief overview of these systems. A much larger survey and bibliography are provided in [13].

Netkit emulates network devices by means of User-Mode Linux instances. The same approach is adopted by VNUML and UMLMON, but they have been conceived with different design goals in mind and therefore rely on a different user interface and architecture.

VNUML [5,6,20] was initially developed to test IPv6 networks in their preliminary deployment stages, while Netkit has been conceived since the beginning as a tool to allow people, even with limited technical skills, to experiment with a large range of networking technologies.

VNUML virtual machines are configured to start a set of standard services by default during the boot phase, including an ssh server and a name server. These services are needed to support some of the features of VNUML. However, this increases the startup time of single virtual machine as well as its memory requirements. Netkit reduces default startup time services in the virtual machines to the very essential, leaving to the users the choice to launch additional services upon demand. As a consequence, a Netkit virtual machine consumes up to 15MB of memory on the host whereas a VNUML virtual machine uses up to 32MB.

VNUML requires that the network to be emulated is described in an XML based language. In our opinion this choice has several drawbacks: users need to learn yet another configuration language, network descriptions may be hard to read and may need specific tools to ease editing, and adding a new technology into the networking emulation system may require extending the language and the parser. On the other hand, the description of a Netkit lab consists of configuration files with standard syntax which may already be familiar to users and require no independent processing, and commands used to control services and apply runtime settings are placed in distinct files and use regular Unix syntax. This approach keeps the Netkit scripts decoupled from the set of supported technologies and makes it possible to easily enrich them, even for the user.

Concerning the ease of installation, VNUML assumes that the user has administrative privileges on the target host. Only recent releases of VNUML tolerate disabling some features to enable running in an unprivileged environment. Conversely, the installation of Netkit is supposed to be performed by a user with standard privileges and augmentation of privileges only takes place when the user explicitly requests to connect a virtual machine to an external network. Moreover, Netkit only relies on standard system tools that are available in most Linux environments, and can be installed by following a very simple procedure that is independent of the target Linux distribution. Instead, VNUML provides facilities that simplify the installation on Debian based distributions, while the standard installation procedure requires users to manually get some additional components (e.g., the virtual hub software) which may not be easy to find and install.

UMLMON [7] is a supervisor that provides tools to configure and manage a pool of virtual machines running on a single host as a solution to perform virtual server hosting and building virtual security zones. Experimentation of networks in a virtual environment is supported as a side effect of this approach. The architecture of UMLMON is based on a Remote Procedure Call interface but also provides utilities to manage virtual machines via a command line or a web interface. A UMLMON agent daemon takes care of actually starting or stopping virtual machines as well as applying the required configurations. UMLMON also provides tools to manipulate disk images for the virtual machines and enforces security by running virtual machines in a chroot jail.

The approach adopted by UMLMON is limited and primarily targeted to system administrators rather than network operators or users willing to learn networking. In fact, a UMLMON user is supposed to build a User-Mode kernel and a filesystem image on his own, as none are provided with the UMLMON packages. Moreover, virtual machine settings as well as the topology of the emulated network are described in a system-wide configuration file that requires the specification of several technical details. Therefore, even the setup of a single virtual machine is not immediate and requires root privileges. Installing UMLMON is not simple either and again requires administrator privileges.

Other approaches known in literature are based on technologies that are uncomfortable to install and manage. This is the case for Einar [30], based on the Xen [15, 24] virtual machines hypervisor, and IMUNES [27], based on an extended FreeBSD kernel running many independent network stacks in user space. Both the projects provide a live CD, while IMUNES also offers the option to be installed on a FreeBSD host. As a consequence, both products require a dedicated machine to run.

Other simulation environments like PlanetLab [12, 34], Emulab [9, 26], and Modelnet [18, 23] are targeted to large scale experimentations and make use of clusters of servers. Some of the PlanetLab nodes are used to run the VINI [1,38] virtual network infrastructure, which exploits User-Mode Linux. However, usage of resources in these environments is subject to approval and often requires the user to be affiliated with an accredited organization, e.g., a research institute. Moreover, it takes some steps before actually earning the right to run an experiment [4,16,17], and emulation software is usually tuned to run on a cluster of machines [23].

A related project is VDE [2, 19], which is a set of tools to create and manage a virtual network that spans a set of arbitrarily distributed physical computers.

To our current knowledge, Netkit is the only environment which combines several features that effectively facilitate experimenting with virtual networks on ordinary hardware, namely: ease of installation, a flexible system for storing and managing labs, no specific language to learn, no need for administrative privileges, and minimum resource consumption, allowing users to efficiently run several tens of nodes on a standard workstation.

8. CONCLUSIONS AND FUTURE WORK

Netkit is a flexible and easy to use environment that provides users with a familiar environment and well known networking software to perform experiments on a wide range of networking technologies. The community of Netkit users is constantly growing: at the time of writing Netkit has about 70 monthly downloads and a mailing list with 133 subscribers.

Netkit scales reasonably well and has been used to emulate real world networks. However, one of the most natural contexts of application of Netkit is probably didactics, also supported by the many ready to use labs available on the Web site along with high quality teaching material. Netkit is effectively exploited within University level networking courses, where it gives the students the opportunity to experiment with the protocols and services they are learning. However, we believe that Netkit may prove itself useful for both operators and researchers in several other contexts, ranging from testing of configurations before deployment to debugging and development of new services and protocols, from studying abnormal routing behaviors to validating theoretical models by experimentation.

The plan for further development of Netkit includes: (i) improvements to the user interface and to the procedure for self testing the labs, to further facilitate the setup of virtual networks, (ii) preparation of further labs that emulate more realistic ISP-like networks, e.g., using MPLS routing or SNMP managed routers, and (iii) integration with VDE [2], in order to support the creation of emulated networks that are distributed across many physical hosts, thus achieving even better scalability. The Netkit community is also supported by the availability of a public repository that collects labs and teaching material proposed by the users. Contributions undergo a revision process that ensures a reasonable quality of the published material. We believe that the opportunity to share emulated experiences will help in growing the Netkit community and in setting up an agile revision process.

Acknowledgments

We acknowledge Giuseppe Di Battista and Maurizio Patrignani for their invaluable contributions in devising and developing Netkit. We would also like to thank Stefano Pettini for his contributions to the ltools, Fabio Ricci for introducing the self test procedure, and Sandro Doro for maintaining the live CD version of Netkit.

9. REFERENCES

[1] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. *ACM SIGCOMM Computer Communication Review*, 36(4):3–14, Sep 2006.

[2] Renzo Davoli. VDE: Virtual Distributed Ethernet. In *Proc. 1st International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2005)*, pages 213–220. IEEE Computer Society, 2005.

[3] Debian. APT Howto. `http://www.debian.org/doc/manuals/apt-howto/`.

[4] Emulab Community. Emulab Documentation: How to get started. `http://www.emulab.net/docwrapper.php3?docname=auth.html`.

[5] Fermín Galán and David Fernández. VNUML: Una Herramienta de Virtualización de Redes Basada en Software Libre. In *Proc. Open Source International Conference 2004*, pages 35–41, Feb 2004. In Spanish.

[6] Fermín Galán, David Fernández, Javier Ruiz, Omar Walid, and Tomás de Miguel. Use of Virtualization Tools in Computer Network Laboratories. In *Proc. 5th International Conference on Information Technology Based Higher Education and Training (ITHET 2004)*, pages 209–214, Jun 2004.

[7] Gerd Stolpmann. UMLMON. `http://www.gerd-stolpmann.de/buero/umlmon.html.en`.

[8] International Computer Science Institute, Berkeley, California. XORP Open Source IP Router. `http://www.xorp.org/`.

[9] Jay Lepreau. Emulab: Recent Work, Ongoing Work. Talk at DETER Lab Community Meeting, Jan 2006.

[10] Jeff Dike. *User Mode Linux*. Prentice Hall, Apr 2006.

[11] John W. Stewart. *BGP4: Inter-Domain Routing in the Internet*. Addison-Wesley, Reading, MA, 1999.

[12] Larry Peterson and Timothy Roscoe. The Design Principles of PlanetLab. *ACM SIGOPS Operating Systems Review*, 40(1):11–16, 2006.

[13] Massimo Rimondini. Emulation of Computer Networks with Netkit. Technical Report RT-DIA-113-2007, Roma Tre University, Jan 2007.

[14] Paolo Giarrusso. UML Utilities. `http://www.user-mode-linux.org/~blaisorblade/uml-utilities/`.

[15] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauery, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proc. 19th ACM Symposium on Operating System Principles (SOSP 2003)*, Oct 2003.

[16] PL-VINI. VINI: Getting Started. `http://www.vini-veritas.net/documentation/pl-vini/user/start`.

[17] PlanetLab Consortium. PlanetLab FAQ: Procedure to get a slice. http://www.planet-lab.org/FAQ.

[18] Priya Mahadevan, Adolfo Rodriguez, David Becker, and Amin Vahdat. MobiNet: A Scalable Emulation Infrastructure for Ad hoc and Wireless Networks. In *Proc. 2005 Workshop on Wireless Traffic Measurements and Modeling (WiTMeMo 2005)*, pages 7–12. USENIX Association, 2005.

[19] Renzo Davoli. VDE: Virtual Distributed Ethernet. http://sourceforge.net/projects/vde/.

[20] Technical University of Madrid (UPM), Telematics Engineering Department. VNUML. http://jungla.dit.upm.es/~vnuml/.

[21] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. The Stable Paths Problem and Interdomain Routing. *IEEE/ACM Transactions on Networking*, 10(2):232–243, 2002.

[22] Timothy G. Griffin and Gordon Wilfong. On the Correctness of IBGP Configuration. *Proc. SIGCOMM 2002*, 32(4):17–29, 2002.

[23] University of California San Diego, Department of Computer Science. ModelNet. http://modelnet.ucsd.edu/.

[24] University of Cambridge, Networks and Operating Systems Group. XEN. http://www.cl.cam.ac.uk/research/srg/netos/xen/.

[25] University of Roma Tre, Computer Networks Research Group. Netkit. http://www.netkit.org/.

[26] University of Utah. Emulab Network Emulation Testbed. http://www.emulab.net/.

[27] University of Zagreb, Department of Telecommunications. IMUNES – An Integrated Multiprotocol Network Emulator/Simulator. http://www.tel.fer.hr/imunes/.

[28] Walter M. Fuertes and Jorge E. López de Vergara. A Quantitative Comparison of Virtual Network Environments Based on Performance Measurements. Poster at the 14th Workshop of the HP Software University Association, Jul 2007.

[29] Debian GNU/Linux. http://www.debian.org/.

[30] EINAR (Einar Is Not a Router) Router Simulator. http://www.isk.kth.se/proj/einar/.

[31] GARR - The Italian Academic and Research Network. http://www.garr.it/.

[32] The Linux Kernel Archives. http://www.kernel.org/.

[33] Packages installed in Netkit filesystem version F3.0a. http://www.netkit.org/download/netkit-filesystem/installed-packages-F3.0a.

[34] PlanetLab Consortium. http://www.planet-lab.org.

[35] Quagga Routing Suite. http://www.quagga.net/.

[36] User-mode Linux Kernel. http://user-mode-linux.sourceforge.net/.

[37] UML Utilities. http://user-mode-linux.sourceforge.net/dl-sf.html.

[38] VINI – A Virtual Network Infrastructure. http://vini-veritas.net/.

[39] Y. Rekhter and P. Gross. Application of the Border Gateway Protocol in the Internet. RFC 1772, Mar 1995.

[40] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, Jan 2006.