A network testbed for commercial telecommunications product testing

Denis Salopek*, Valter Vasić*, Marko Zec*, Miljenko Mikuc*, Mladen Vašarević†, Vladimir Končar†,
*University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia
denis.salopek@fer.hr, valter.vasic@fer.hr, zec@fer.hr, miljenko.mikuc@fer.hr
†Ericsson Nikola Tesla, Zagreb, Croatia
mladen.vasarevic@ericsson.com, vladimir.koncar@ericsson.com

***Abstract*—Have open-source network topology emulators outgrown the realm of academic and educational playgrounds? We tackle with that question by dissecting our experiences with preproduction testing of functional blocks implemented in commercial carrier-grade telco equipment, which we performed in a testbed based on open-source tools. As the concepts and mechanisms on which popular network emulation platforms are based vary, so does their applicability to different problem domains in telco product testing. Our survey of the existing open-source tools reveals that the spectrum of their features and limitations is multidimensional, with the choice of virtualization techniques applied being crucial to the raw packet processing throughput, topology size scaling, experiment instantiation speeds, and the flexibility of integrating diverse tools in a single testbed environment. We measured key performance metrics for two conceptually similar emulation platforms (IMUNES and CORE) running on different operating systems (FreeBSD and Linux). Three testbed scenarios for commercial telecommunication products were described and analyzed.**

***Index Terms*—Network emulation, Virtualization, Internetworking, Software Routers**

1. INTRODUCTION

The term "network emulation" usually refers to the ability of a controllable, synthetic environment to efficiently interact with real computer networks and traffic in real time. On the other hand, network simulation systems are focused on modeling various communication phenomena on either micro or macro scale in entirely isolated synthetic environments which have its own notion of (usually virtual) time. In many cases such tools may be extended to permit interaction with real networks, although typically subject to various restrictions, significant overheads and thus reduced throughput compared to their dedicated emulation counterparts.

There is also considerable ambiguity in literature on what functionality is expected to be found in "network emulation" tools. The tools which emerged during 1990s [1] [2], some of which are still actively maintained today such as [3], focused on emulating properties of network links and queues, such as propagation and queuing delays, bandwidth constraints, packet loss etc. While such tools were categorized as "network emulator", they were capable of emulating only a single network link (with multiple parallel queues). Advances in various virtualization technologies applicable to commodity hardware and operating systems [4] [5] [6] [7] from late 1990s and early 2000s paved the way for new tools which permit complex network topologies to be emulated on a single machine with various degrees of efficiency.

In this article we focus on network topology emulation tools, particularly those based on open-source components, and their application to testing of functional blocks in commercial telecommunications equipment. Publications covering application of open-source network testing tools in commercial environments are in a surprisingly short supply, which was our main motivation to document first-hand experiences we gained in this area.

The rest of the article is organized as follows. In related work section we present a brief overview of existing network topology emulation tools. Section 3. provides an assessment of key performance metrics for the emulation platform of our choice and comparison with a similar counterpart running on a different operating system. In section 4. we describe three different testbed scenarios through which we illustrate and discuss our choices of applied virtualization techniques and test software design decisions. Our findings and directions for future work are summarized in the concluding section.

2. RELATED WORK

A recent reasonably comprehensive and accurate survey [8] provides an overview of the majority of contemporary network emulation tools, so here we attempt only to briefly summarize their main features and categorize them based on the following criteria:

Communication overhead: As packets flow through an emulated network it may be necessary to copy them from one isolation context (representing a virtual node) to another, or from the operating system (OS) kernel to userspace and vice versa, depending on internal architecture of each tool. Copying packet payloads and switching execution context during packet handoff can incur significant performance penalties and introduce undesirable latencies and jitter. Copy operations required to move a packet from one virtual node to another are reported in the first column of Table I.

Node virtualization: The illusion of multiple independent network nodes running on a single physical machine can be accomplished via different virtualization methods, which are outlined in second column of Table I. Full machine virtualization platforms used by network emulators include QEMU, VirtualBox or VMware, which provide the highest

Table I
OVERVIEW OF NETWORK EMULATION TOOLS

Tool name	Communication overhead	Node virtualization	Filesystem virtualization	Experiment startup speed	Link emulation	GUI
IMUNES [9]	zero copy	FreeBSD jails	Layered (unionfs)	Very fast (pipelined)	Point-to-point	Yes
CORE [10]	2 copies (Linux) 0 copy (FreeBSD)	LXC containers / FreeBSD jails	None (shared filesystem)	Fast (sequential)	Multipoint (WLAN)	Yes
Mininet [11]	zero copy	Network namespaces	None (shared filesystem)	Fast (sequential)	Point-to-point	No
MLN [12]	two copies	Full virtualization / Paravirtualization	Independent or COW disk images	VMs' boot and shutdown time	None	No
Marionnet [13]	two copies	Paravirtualization	Independent or COW disk images	VMs' boot and shutdown time	Point-to-point	Yes
Cloonix [14] Virtualsquare [15]	two copies	Full virtualization / Paravirtualization	Independent disk images	VMs' boot and shutdown time	Point-to-point	Yes
Netkit [16]	two copies	Full virtualization	Independent or COW disk images	VMs' boot and shutdown time	None	No
GNS3 [17] VNX [18]	two copies	Full virtualization	Independent disk images	VMs' boot and shutdown time	None	Yes

degree of isolation between virtual nodes but at a high I/O performance cost and a significant memory footprint, which combined limit the scalability to only a handful of virtual nodes on a single physical machine. Paravirtualization platforms such as Xen [4], KVM or UML which attempt to amortize some of the high I/O virtualization costs are also employed by certain network emulation tools. Other emulation tools leverage OS compartmentalization techniques such as FreeBSD jails, LXC containers or OpenVZ, combined with network stack virtualization [7] or network namespaces, which provide an illusion of multiple isolated execution environments with private networking state on a single OS image with negligible packet I/O overhead.

Filesystem virtualization: Network emulation tools relying on both full system virtualization and paravirtualization typically require independent filesystem images for each virtual node. This significantly contributes to their high memory footprint and I/O overhead associated with running processes in each emulated node, where only I/O overhead may be moderately reduced by using a single "master" disk image combined with copy-on-write images for each emulated node. Emulation tools which leverage OS compartmentalization techniques may more economically virtualize file access by using copy-on-write filesystems such as ZFS, or by using layered filesystems such as UNIONFS [19] which effectively emulate copy-on-write semantics but with further reductions in memory footprint for typical network-centric workloads. Finally, in an attempt to minimize the memory footprint of each virtual node, some emulation tools do not provide any means for filesystem virtualization, which requires many applications to be modified to run properly in multiple instances, while opening a wide vector for interference between processes running in separated virtual nodes.

Startup speed: Tools which leverage containers / jails instantiate emulated network topologies quickly (several nodes per second), while paravirtualized virtual machines take longer to start, and for fully virtualized nodes instantiation delays can stretch to minutes per node. Those delays depend on the guest OS running in a virtual node, as well as on the total number of nodes in an experiment. Pipelined instantiation of virtual nodes, which spreads node startup and initial configuration tasks on multiple CPU cores, improves experiment startup delays compared to tools which (unnecessarily) strictly serialize node instantiation.

Link emulation: The last column indicates whether a tool is capable of emulating link impairments such as bit-error-rate (BER), delay, bandwidth etc. Feature sets and scalability of impairments emulation vary between tools, but in general their accuracy (timing resolution) and jitter is influenced by the system load and OS scheduler behavior, which in real-world workloads limit the precision to around 1 ms range. Among all the reviewed tools only CORE provides a

dedicated module for emulating multi-point wireless media.

GUI: a graphical user interface simplifies topology specification and manipulation, thus lowers the entry barrier for users not trained in UNIX system administration, which comes handy especially in educational applications.

3. Performance considerations

Driven by the requirement for supporting experiments with moderately high packet rates (around 100.000 packets / second with zero packet loss), our choice of emulation platforms was narrowed down to those leveraging OS compartmentalization techniques which allow for packets to traverse virtual network topologies with minimum overhead. Our decision to use FreeBSD as the base OS with IMUNES as the experiment control and management plane was initially driven primarily by previous in-house experiences and developer availability, though the advantages of the BSD licensing model also become apparent early in the platform adoption process as the team was relieved from legal burdens when developing and exchanging kernel patches and extensions without releasing them to the outside world, which would otherwise require following rigorous and exhausting approval processes. Later on, as more testing teams got involved in the project, the ability to quickly instantiate and restart experiments was identified as important (though not critical), so we decided to include the related benchmarks in this section.

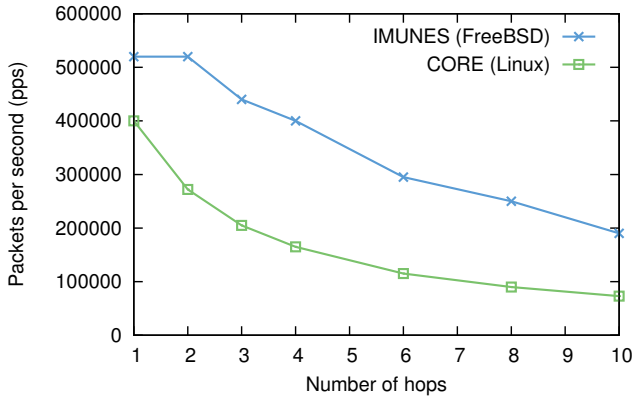

Figure 1. IPv4 traffic forwarding throughput for IMUNES (FreeBSD) and CORE (Linux). IPv4 fast forwarding option was enabled on FreeBSD, while reverse-path filtering was disabled on Linux in order to reduce the number of IPv4 lookups per packet.

As our main concern was emulation platform's packet forwarding capacity, in the first synthetic test we measured loss-free forwarding rate of our reference machine (Intel Xeon E5-1650 CPU, 6 cores at 3.20 GHz, 8 GB of RAM) while varying the length of a chain of virtual routers through which externally generated traffic was injected. We used FreeBSD 9.3-RELEASE (amd64) as the base OS throughout the tests. For comparison we included throughput figures of running identical tests on Linux (Ubuntu 14) using CORE as the emulation control plane. Traffic was generated and measured using netmap-based tools [3] running on an external machine, which was connected to the emulation server via a dual-port Intel X520-SR2 10G Ethernet interface card. The results are shown in figure 1. The aggregate throughput (packet rate times number of hops traversed) of FreeBSD-based setup peaks just above 2 million packets per second for a router chain with eight nodes, while Linux / CORE results are considerably lower, which can be attributed to the fact that CORE uses a user-space daemon for emulating link-level impairments, while with FreeBSD / IMUNES the entire forwarding path is contained inside the kernel, hence packet copy operations are avoided. Moreover, we observed that packet forwarding load on FreeBSD (mainly netgraph [20] worker threads) was reasonably well balanced between available CPU cores, while on Linux / CORE a single user-level thread responsible for emulating link-level impairments stood out as an apparent performance bottleneck, which indicates that with careful datapath rearchitecting the performance gap between Linux / CORE and FreeBSD could be reduced. We also detected that at higher traffic rates the Linux / CORE setup could not forward traffic with zero packet loss, while such a problem could not be observed on FreeBSD. Once overloaded with inbound traffic, the forwarding throughput of both FreeBSD and Linux / CORE would collapse, although operating systems would not livelock, i.e. they both remained controllable and reachable via a separate network interface.

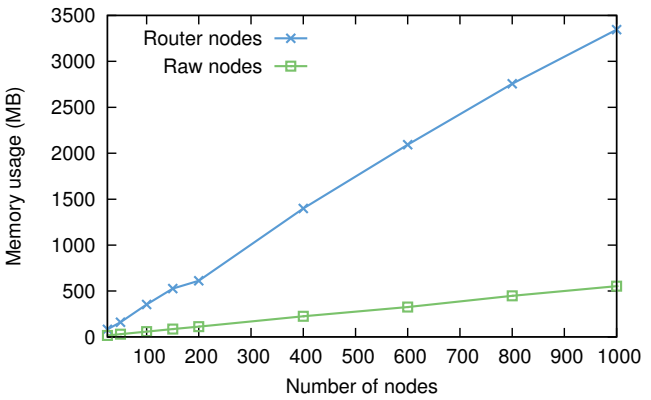

Figure 2. Memory footprint as a function of experiment size in IMUNES. "Raw" nodes are empty containers with no user-level processes running. "Router" nodes run quagga routing daemon and actively exchange IPv4 and IPv6 routes via RIP.

Figure 2 shows that baseline memory usage in IMUNES is fairly small, around 600 KB per instantiated virtual node with no running processes, and around 3500 KB when running quagga routing daemons. This permits sizable experiment topologies to modelled, even with each virtual node having a private copy-on-write (COW) view of filesystem hierarchy (unlike in CORE or Mininet).

On our test machine IMUNES was capable of instantiating between 7 and 30 virtual nodes per second, depending on

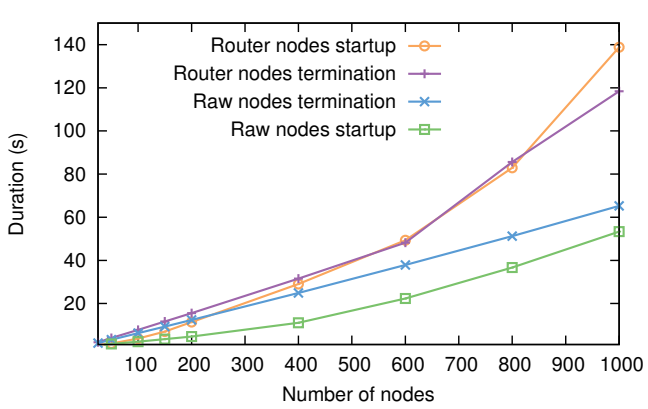

Figure 3. The impact of experiment size on startup and termination duration in IMUNES / FreeBSD. "Raw" nodes are empty containers with no user-level processes running. "Router" nodes run quagga routing daemon and actively exchange IPv4 and IPv6 routes via RIP.

experiment size and type of the workload. Experiment termination speed is in the same range, as shown in figure 3. The obtained performance levels are between 4 and 8 times faster than with CORE / Linux using identical topologies on the same test machine, which comes somewhat surprising given that both emulation platforms utilize similar OS virtualization techniques, and given that IMUNES has an additional overhead of creating a private COW filesystem environment for each virtual node, while CORE does not. We attribute the difference in experiment instantiation speed to the fact that IMUNES attempts to spread the load of virtual node creation to multiple threads, while the corresponding tasks in CORE are serialized.

4. Use cases

In this section we present three testbed scenarios that were intensively used in Ericsson Nikola Tesla to test the Cello Packet Platform (CPP) [21] based products, which illustrate the feasibility of replacing complex and expensive physical testbeds with a network emulator.

Emulation enables replacing real hardware, e.g. routers, switches and hosts. This greatly simplifies and speeds up testbed setup and configuration because there is no need to manually configure and connect a large amount of different physical equipment. The setup is lightweight and portable, thus it can be easily relocated to another testing site. Therefore both the testing equipment and variable expenses are reduced. This decreases the total cost of the product and makes it more competitive in the market.

4.1 IPsec peers testbed

In this testbed, it was necessary to simultaneously support 25 different IPsec gateways (Internet Key Exchange [22] peers) connected to Ericssons CPP products as shown in Figure 4. The following was needed for the testbed:

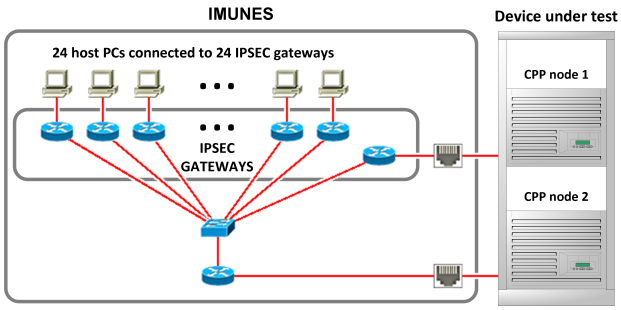

Figure 4. IPsec peers testbed

- 25 different IPsec gateways. One of the gateways is used for communication between two CPP nodes.
- 24 hosts with different IP addresses, each connected to its own IPsec gateway.
- 1 router.
- 1 switch.

To carry out this testbed using real hardware, one or more physical IPsec gateways with 25 different ports (IP addresses) and one or more physical hosts with 24 different ports were needed. All of this was run in a single IMUNES experiment because all nodes have a different instance of the network stack [7] and an independent execution environments [6].

This experiment contains three different types of emulated nodes. The switch is a netgraph [20] node that does packet switching in the kernel. Host (raw) nodes are persistent jails without any running applications with a separate interface and network stack. Router node is a persistent jail (same as a host) which is running a Quagga [23] routing daemon and supports RIP, RIPng, OSPFv2 and OSPFv3. IPsec gateway nodes are routers that are running the Strongswan IPsec utilities [24] that provide all required functionalities needed for the testbed. Physical network interface cards (NIC) can be assigned to nodes in the emulated environment and connect emulated topologies with real-world equipment.

The average startup and termination times for this testbed are respectively 1.7 seconds and 2.9 seconds. The average amount of memory needed for this testbed is 88 MB.

4.2 IPv6-to-IPv4 translation testbed

In testbed shown in Figure 5, various combinations of IPv6 / IPv4 network configurations were tested. This scenario needed the following:

- Virtual LAN (VLAN) termination support on at least 2 links for both the data and control planes.
- Generic Tunnel Interface (GIF) termination for the data plane (mobile call communication data) for at least 10 different IP addresses - around 100 Mbit/s in total.
- IPv6-to-IPv4 translation support for control plane data (handles call establishment and termination) - 7 Mbit/s of SCTP data running through NAT64.

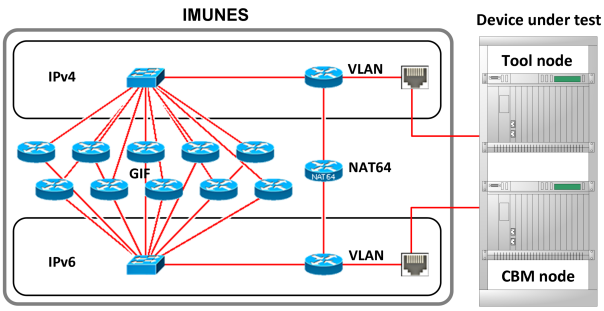

Figure 5. IPv6-to-IPv4 translation testbed

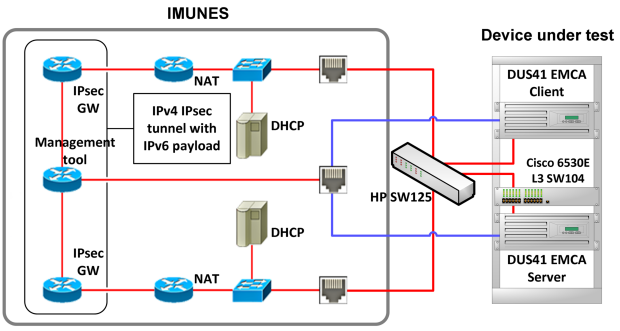

Figure 6. IPsec testbed

This testbed, deployed using real hardware, would require a router for GIF tunnel termination, 2 switches for VLAN termination and a router for IPv6-to-IPv4 translation. Besides that, it was necessary to run three of these experiments at the same time which would require three times more equipment, but that was easily emulated on a single machine because IMUNES has integrated support for running multiple experiments at once.

Emulated topologies can be complemented by scripts that simplify topology configuration and deployment. A dedicated set of tools is included in the standard distribution that enables node and link manipulation. VLAN and GIF termination was done on emulated router nodes which were modified by a single external script that configured VLAN and GIF interfaces. The NAT64 node is an extended router node that automatically configures Tayga [25], a NAT64 userspace implementation for Linux, according to the configuration specified in the GUI.

The average startup and termination times for this testbed are around 1.0 second, while the average memory footprint is 22 MB.

4.3 IPsec testbed

The IPsec testbed is shown in Figure 6. It was used to test CPP products (DUS41 Client and Server). The requirements for the test environment were as follows:

- IPsec gateways that encapsulate IPv6 client payloads in an IPv4 IPsec tunnel.
- DHCP servers that assign DHCP addresses to newly connected clients.
- NAT daemons that translate traffic that is communicated between nodes and IPsec security gateways.
- Management tool to connect and manage the device under test (DUT).

The minimum hardware requirements for this testbed would be 1 IPsec gateway, 1 router with NAT and DHCP support, 1 DUT management node and 2 switches.

For this testbed, additional tools were integrated into virtual nodes (illustrated in Figure 6). Host nodes were extended by adding an ISC DHCP server [26] to provide address assignment (unmodified FreeBSD binaries can run virtual nodes). Router nodes were configured to run FreeBSD NAT daemons in combination with IPFW firewall support. IPsec gateways were the same as in the first testbed. Finally, the Management tool node was configured to run custom management software to avoid the addition of another physical machine to the testbed. The management tool usually runs on Linux and was integrated into the emulated router node (running in the IPv4 part of the network).

The average startup and termination times for this testbed are around 0.7 seconds, whereas the memory footprint is around 42 MB.

5. Conclusion

The described experiences with using open-source software for constructing network testbeds have shown that such tools offer excellent flexibility for adapting to very specific requirements of various test scenarios, provided that the tools are not pushed beyond their known limitations, with packets per second throughput being the most obvious obstacle. The ability to efficiently run multiple experiments in parallel on a single emulation server turned out to be pivotal for supporting multiple testing teams working simultaneously on different problem sets. The choice of OS compartmentalization, as currently the most efficient way of multiplexing many virtual nodes on a single physical machine, have proved sufficient for our test scenarios, some of which required loss-free throughputs in excess of 100.000 packets per second. As it is unrealistic to expect that packet per second throughputs of standard data planes in commodity operating systems are going to significantly improve in the foreseeable future, we are exploring new approaches for supporting applications which will require orders of magnitude higher throughputs (10G and 40G Ethernet). New software abstractions for efficient packet handoff such as netmap [3] offer the throughput headroom, but require packet processing datapaths to be constructed from scratch, which calls for long-term development efforts without inherent guarantees for success.

REFERENCES

[1] M. Allman, A. Caldwell, and S. Ostermann, "One: The ohio network emulator," tech. rep., 1997.

[2] M. Carson and D. Santay, "Nist net: a linux-based network emulation tool," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 111–126, 2003.

[3] L. Rizzo, "Portable packet processing modules for os kernels," *Network, IEEE*, vol. 28, pp. 6–11, March 2014.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," vol. 37, pp. 164–177, ACM, 2003.

[5] C. A. Waldspurger, "Memory resource management in vmware esx server," vol. 36, pp. 181–194, ACM, 2002.

[6] P.H. Kamp and R.N.M. Watson, "Jails: Confining the omnipotent root," in *Proceedings of the 2nd International SANE Conference*, vol. 43, p. 116, 2000.

[7] M. Zec, "Implementing a clonable network stack in the freebsd kernel.," in *USENIX Annual Technical Conference, FREENIX Track*, pp. 137–150, 2003.

[8] E. Lochin, T. Perennou, and L. Dairaine, "When should i use network emulation?," *annals of telecommunications-annales des télécommunications*, vol. 67, no. 5-6, pp. 247–255, 2012.

[9] M. Zec and M. Mikuc, "Operating system support for integrated network emulation in IMUNES," in *1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)*, p. 3, 2004.

[10] J. Ahrenholz, C. Danilov, T. Henderson, and J. Kim, "Core: A real-time network emulator," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pp. 1–7, Nov 2008.

[11] B. Lantz, B. Heller and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *9th ACM SIGCOMM Workshop on Hot Topics in Networks*, p. 19, ACM, Oct 2010.

[12] K. M. Begnum, "Managing large networks of virtual machines.," in *LISA*, vol. 6, pp. 205–214, 2006.

[13] J.-V. Loddo and L. Saiu, "Marionnet: a virtual network laboratory and simulation tool," in *First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2008.

[14] D. Rehunathan, S. Bhatti, V. Perrier, and P. Hui, "The study of mobile network protocols with virtual machines," in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, (ICST, Brussels, Belgium, Belgium), pp. 115–124, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.

[15] R. Davoli and M. Goldweber, "Virtual square (v 2) in computer science education," in *ACM SIGCSE Bulletin*, vol. 37, pp. 301–305, ACM, 2005.

[16] M. Pizzonia and M. Rimondini, "Netkit: easy emulation of complex networks on inexpensive hardware," in *Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*, p. 7, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[17] Y. WANG and J. WANG, "Use gns3 to simulate network laboratory," *Computer Programming Skills & Maintenance*, vol. 12, p. 046, 2010.

[18] D. Fernandez, A. Cordero, J. Somavilla, J. Rodriguez, A. Corchero, L. Tarrafeta, and F. Galan, "Distributed virtual scenarios over multi-host linux environments," in *Systems and Virtualization Management (SVM), 2011 5th International DMTF Academic Alliance Workshop on*, pp. 1–8, Oct 2011.

[19] J.-S. Pendry, U. Sequent, and M. K. McKusick, "Union mounts in 4.4 bsd-lite," *AUUGN*, p. 1, 1997.

[20] A. Cobbs, "All about nethgraph." http://people.freebsd.org/~julian/netgraph.html.

[21] L. M. L. Kling, . Lindholm and G. B. Nilsson, "CPP - Cello packet platform," tech. rep., Ericsson, 2002.

[22] Y. N. C. Kaufman, P. Hoffman and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)." RFC 5996 (Proposed Standard), Sept. 2010.

[23] "Quagga project, quagga routing suite." http://www.quagga.net.

[24] A. Steffen, "strongSwan - the OpenSource IPsec-based VPN Solution," Nov. 2014.

[25] N. Lutchansky, "TAYGA Simple, no-fuss NAT64 for Linux."

[26] I. S. Consortium, "ISC dhcpd server," 2014.