A Virtual Integrated Network Emulator on XEN (viNEX)

Abraham Mukosi Mukwevho* †
School of Computing
University of South Africa
P O Box 392, UNISA, 0003
mukosi@gmail.com

John Andrew van der Poll
School of Computing
University of South Africa
P O Box 392, UNISA, 0003
vdpolja@unisa.ac.za

Robert Mark Jolliffe
bobjolliffe@gmail.com

ABSTRACT

The recent progress on virtualization technologies has made it possible to deploy multiple hosts instances with operating systems running real network protocol stacks on one single server. The objective of this paper is to explore whether it is feasible to use such environments for network emulation and simulation. Some significant amount of research is taking place in this area, this includes Emulab [6] virtualization, and IMUNES [12] system. Both Emulab and IMUNES are based on FreeBSD Jails.

Very little is known about using traditional virtualization platforms (such as Xen and VMware) for virtual emulators. As part of our research, we will attempt to develop a virtual emulator (viNEX[1]) based on Xen. Having identified the limits and weaknesses of this approach, we also propose some areas where viNEX can be useful.

Categories and Subject Descriptors

D.4.8 [**Performance**]: Measurements; I.6.7 [**Simulation Support Systems**]: Environments

General Terms

Network Simulation and Emulation

Keywords

Computer networks, Simulators, Emulators

*Abraham Mukwevho is a M.S student and primary author of this paper at the University of South Africa.

†Configuration and Administration **scripts** mentioned in this paper can be accessed online at - **http://sites.google.com/site/mukosi/**

[1]from now onwards, **viNEX** will be used as a short for **V**irtual **I**ntegrated **N**etwork **E**mulator based on **X**EN

1. INTRODUCTION

Network research experiments have traditionally been conducted in an *emulated* or *simulated* environment. Emulators (such as Emulab [6]) are normally based on physically deployed networks which are associated with high procurement and maintenance costs, complex configurations and infrastructure (servers, routers and gateways). On the other hand, network simulators such as NS-2 [15] provide a self-contained and simple environment that can be hosted on a single host. Simulators provide a synthetic environment which is only an approximation of the real world and therefore the results might not be a true reflection of real world. Furthermore, network protocol components developed in a simulated environment require a significant amount of code refactoring in order to migrate them into the real world. This is mainly because simulated environments do not run real network protocol stacks, instead they use software modules that mimic real world protocol stacks. It is also possible to combine both emulation and simulation in one environment, for example; in Emulab simulation can be provided by instantiating NS-2 traffic generators or sinks on one of the topology nodes. To overcome limitations associated with simulators, emulators provide an alternative approach whereby network protocols can be developed while interacting with real protocol stacks, and hence eliminating the need to migrate protocol code to the real world. Network emulators have traditionally been based on physical network deployments (a good example is the original Emulab). The recent advances on the development of virtualization technologies has now made it possible to deploy multiple hosts on one single environment and interconnect them to provide a complete network environment. These virtual hosts run real network protocol stacks and therefore provide an emulated environment that can be used for network research experiments.

Our fundamental goal is to explore the possibility of using a traditional virtualization platform like Xen to build a stand-alone network emulator hosted on one single server or PC. Part of the rationale for this research is to be able to create a freely distributable experimental environment for use, for example, by distance education students who don't have computer laboratory access. We are also aware that the software bridges will be a performance hit. What we don't yet know is how slow will be too slow. Furthermore, the viNEX environment is never going to be useful for high performance fast network emulation, but it might still be useful for other educational scenarios.

Our emulator (viNEX) was built using free and open source software. The use of open source software in networking re-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5745
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5745

search continues a very long tradition, this includes NS-2, FreeBSD Jails, and Emulab. We selected Xen as the virtualization platform because at the time the research was initiated Xen was the most viable open source platform you could run "any OS on". Despite our focus on building mini-nodes using NetBSD, the system is not, and is not meant to be, restricted to using NetBSD nodes. In other words the ability to run any OS is part of our high level design goals. Other open source technologies used include; FreeBSD 7.0 and NetBSD 4. NetBSD is used to implement the experiment topology nodes. FreeBSD is used to provide traffic shaping and link emulation using Dummynet and IPFW which are also open source technologies.

We would also like to make a special note to our audience - please note that our emulator is work-in-progress therefore it is by no means in a complete status, it an ongoing research work and we are continually improving it. On the other hand, we are aware of some limitations of this approach, we will be addressing some of them as part of the research. All scripts and progress work on viNEX can be referenced online at [13]

The rest of this paper is structured as follows. In Section 2, we give background work in support of this research. The original contribution work (viNEX) is described in Section 3 through to Section 4. We approach the conclusion of this paper by looking at current and other related research work at Section 5. We conclude this paper and give pointers to future research work in Section 6.

2. BACKGROUND

To begin, we provide some overview on *emulation* and *simulation* to form the foundation work for viNEX. Due to the limit in scope for this paper, we could not provide a complete background on Xen networking. A significant amount of technical writing on Xen exists, interested readers can refer to [19], [5] and [24].

2.1 Network Emulation and Simulation

Network research environments can be classified into three categories, i.e. testbed, network simulation and network emulation. A network *testbed* is a physically deployed and configured environment dedicated for conducting network research experiments. It is formed by real networking elements such as end hosts, routers, links (cables), bridges and switches. Some good examples of testbeds include; Emulab [6], and PlanetLab [4]. Both these environments consist of a set of physical servers deployed in a laboratory environment interconnected by switches. Network experimenters normally access these shared testbed environments over the Internet to setup and execute their experiments, results are obtained by downloading captured log files. Some advantages of testbed environments include; experiments are executed in real time and interacting with real protocol stacks deployed at the end hosts and routers. Testbed environments are not dynamic and have a lot of drawbacks; they are difficult to setup, configuration can be tedious and time consuming, physical hardware is extremely expensive to procure, hardware logistic and storage space problems. Furthermore, although offering a real world environment, conducting experiments on testbed offers an uncontrollable, unpredictable and non-repeatable environment.

Network *simulators* are normally implemented as a collection of software modules providing a synthetic network experiment environment. Simulators achieve this by defining and modeling network behavior through the abstraction of network elements, this include a virtual simulated time, and a discrete network events system for traffic generators. Simulators are normally deployed and executed on a single host. Despite the limited emulator functionality of NS-2 [15], NS-2 is a classical and popular example of a network simulator. Network simulators offer a repeatable and controlled environment for experiments. Simulators are easy to setup and configure, and a result, they offer a lot of control to experimenters making them an ideal choice for rapid protocol prototyping and evaluation.

Network *emulation* refers to a hybrid technique that leverages on the features and benefits of both testbed and simulated techniques. Emulation combines the real network elements of the testbed approach to the synthetic or simulated elements of simulation. In most cases, simulated elements of an emulated environment include - network links and intermediate nodes.

Emulab, despite being a physical testbed environment, is a good example of an emulated environment. In Emulab, a transparent FreeBSD delay nodes are inserted between topology links in order to simulate the network boundary conditions using the Dummynet module. NS is also an example of a limited network emulator. NS has recently introduced some limited emulation functionality whereby real network traffic can be subjected to emulated network components. The emulation facility of NS is described in detail at [7].

3. IMPLEMENTATION OF VINEX

The following sections give a technical description of the emulator (viNEX) and its implementation details. Development of viNEX was conducted on a single Linux host, see Table 1 for environment hardware and software configurations;

Table 1: viNEX Development Environment

Operating System	*CentOS 5.1*
Memory	*1 GB*
CPU	*Intel Core 2 Duo, 3.0 GHz CPU with vT Support*
Other Software	*Xen 3.2, NetBSD 4, FreeBSD 7 (with Dummynet and IPFW enabled)*

3.1 Architecture and Design

Figure 1 depicts the high-level architecture of viNEX. The main components of viNEX include: *control network*, *experiment topology nodes*, *traffic shaping node*, *network links*, and testbed *configuration and management scripts*.

3.1.1 Control network

Similar to Emulab [6], a separate *control network* is created to allow users direct access to the experiment *nodes* from within Domain 0. The *control network* is used by setup scripts for access to the nodes in order to configure them for networking by executing commands using SSH. Each topology node (Node X) is assigned a Class C IP address 196.30.225.X for the control network.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5745
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5745

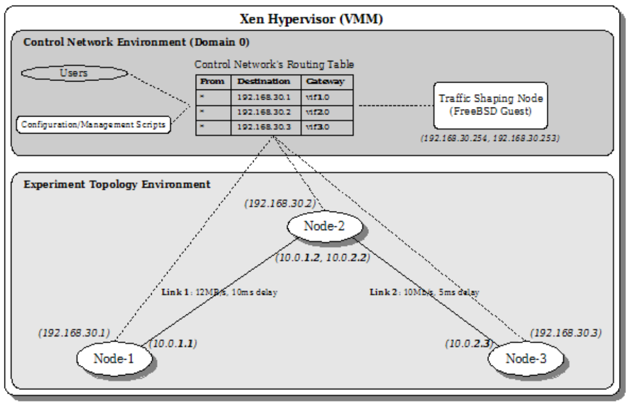
Xen Hypervisor (VMM)
Control Network Environment (Domain 0)
Users
Configuration/Management Scripts
Control Network's Routing Table
From | Destination | Gateway
* | 192.168.30.1 | vif1.0
* | 192.168.30.2 | vif2.0
* | 192.168.30.3 | vif3.0
Traffic Shaping Node (FreeBSD Guest)
(192.168.30.254, 192.168.30.253)
Experiment Topology Environment
(192.168.30.2)
Node-2
(10.0.1.2, 10.0.2.2)
Link 1: 12MB/s, 10ms delay
Link 2: 10Mb/s, 5ms delay
(192.168.30.1)
(10.0.1.1)
Node-1
(10.0.2.3)
(192.168.30.3)
Node-3

Figure 1: High-Level Testbed Architecture

3.1.2 Experiment topology nodes

These nodes form the topology to be used for conducting a network experiment. They are standard Xen HVM guest nodes. All experiment nodes are NetBSD 4 nodes running a minimal kernel.

3.1.3 Traffic shaping node

The traffic shaping node is a FreeBSD node configured as a transparent gateway between network links. In addition to link modeling, the traffic shaper is used to model network boundary condition such as: bandwidth limitation, packet delay, and random packet loss.

3.1.4 Network Links

Links are used to model communication between any pair of experiments topology node. Links are defined inside the traffic shaper node. The traffic shaper node uses a combination of software bridging together with VLANs in order to model the link between two nodes. Dummynet and IPFW are used for bandwidth and delay simulation.

3.1.5 Configuration scripts

This is provided through a collection of Linux shell scripts as follows. All these scripts can be obtained online at [13]:

`start-gateway.sh` - is used to boot the FreeBSD traffic shaping node.

`start-node.sh` - is used for starting any NetBSD experiment topology node as required

`create-link.sh` - is for for creating links between each pair of nodes as specified by the experiment.

`modify-link.sh` - is used to alter the link properties after it has been created.

3.2 Network Links

We now expand and discuss the lower level details of the network link abstraction between any pair of topology nodes. For the purpose of this discussion, please assume the two node topology depicted in Figure 2 below;

A *link* between any pair of virtual nodes is formed by the following network elements; *front-end interfaces, back-end interfaces, vlan subinterfaces, bridges, ebtables rules, ipfw rules and dummynet pipes.* A sample Xen configuration for the scenario in Figure 2 is shown in Listing 1. The important information to note here is the IDs assigned by Xen to each node.

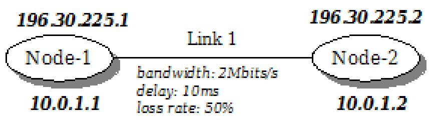

Figure 2: A basic two-node network topology

```
[root@mukosi experiment]# xm list
Name      ID   Mem   VCPUs  State    Time(s)
Domain-0   0  1425     2    r-----    102.0
Gateway    1   512     1    -b----     31.7
Node-1     2    16     1    -b----     12.4
Node-2     3    16     1    -b----     13.1
```

Listing 1: Xen configuration of Figure 2 topology

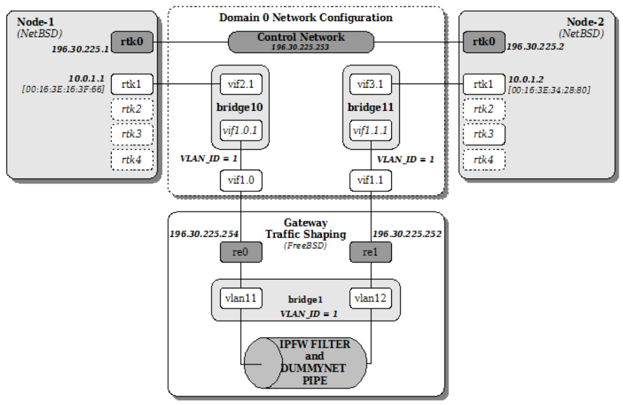

Figure 3: Components of a link connection between two nodes

The following paragraphs briefly describe each network link element as identified above:

3.2.1 Front-end interfaces

These are interfaces running inside each domain. Since we are using HVM, all domains are running the native unmodified network drivers. All interfaces involved are depicted in Figure 3. Each node is allocated *five* front-end interfaces, they configured during the node startup process. The lowest interface, `rtk0` is always reserved for the control network and it is automatically assigned the control network IP address during startup. The IP address allocation scheme for the control network is such that for each Node X, interface `rtk0` is allocated a Class C IP address 196.30.225.X.

3.2.2 Back-end interfaces

They are interfaces inside Dom0 and directly connected to front-end interfaces inside the topology domains as well as the Gateway node. Looking at Figure 3, Node-1 and Node-2's front-end interfaces are directly connected to back-end interfaces `vif2.1` and `vif3.1` respectively. Similarly, the Gateway node's front-end interfaces `rtk0` and `rtk1` are connected to back-end interfaces `vif1.0` and `vif1.1` respectively.

3.2.3 Bridges

For each link, two Linux software bridges are created. The purpose for the bridges is to connect the traffic from the topology nodes directly to the Gateway node for traffic shaping in a protocol independent manner. Packets are forwarded based on Ethernet address and not IP address. Both the node's back-end interface and the Gateway's vlan subinterfaces are joined together to allow traffic routing at layer 2, as a result all protocols can be carried across the links.

3.2.4 VLAN Sub-Interfaces

The Gateway node only has two fixed interfaces (`vif1.0` and `vif1.1`) connecting it directly to the Dom0. Since all links have to go through the FreeBSD Gateway node for traffic shaping, we had to derive a mechanism that will allow the sharing of these two fixed back-end and front-end interfaces among all links. For each link X in the topology, a corresponding VLAN with VLAN_ID = X is created in order to isolate the link's traffic. For the example in Figure 2, a VLAN with VLAN_ID = 1 is defined for Link 1 (see Figure 3).

3.2.5 Ebtables Rules

Ebtables [20] is a Linux packet filter that enabled us to intercept bridged traffic at layer 2 and be able to BROUTE them. Packets are forwarded at layer 2 without having to be passed to layer 3 for routing. The PREROUTING chain of the Ebtables NAT table is used to perform the MAC address translation of the destination using the `dnat` instruction. The destination MAC is changed to the MAC address of the directly connected destination as defined by the topology of the experiment before the packet is passed into the traffic shaper Gateway. Using the basic experiment in Figure 2, for each link, two Ebtables rules are created inside the NAT PREROUTING table by the link configuration script - see Listing 2 for details.

```
1 [root@mukosi experiment]# ebtables -t nat -L
2 Bridge table: nat
3 Bridge chain: PREROUTING, entries: 2, policy: ACCEPT
4 -i vif2.1 -j dnat --to-dst 0:16:3e:34:28:80 --dnat-target ACCEPT
5 -i vif3.1 -j dnat --to-dst 0:16:3e:16:3f:66 --dnat-target ACCEPT
6 ...
```

Listing 2: Ebtables rules for the topology of Figure 2

The two Ebtables rules are listed in line 5 and 6 of Listing 2. The rule in line 5 simply translates the MAC address of any frame that arrived through back-end interface `vif2.1` and set it to the MAC address of the front-end interface of Node-2 (`00:16:3e:34:28:80`) so that the frame can be passed directly after being traffic shaped by the Gateway. Similarly, the rule in line 6 is used to translate the MAC address of any frame arriving directly from Node-2 through interface `vif3.1` and set it to the MAC address of the front-end interface of Node-1 (`00:16:3e:16:3f:66`).

3.2.6 IPFW Rules

Packet filtering is specified using a set of rules that are created by using the IPFW command line utility of FreeBSD. See Listing 3 for the list of IPFW rules. Dummynet pipes are also created using the `ipfw` command line.

```
Gateway# ipfw show
00800  27  1260 pipe 1 ip4 from any to any via vlan11 layer2
00900  27  1260 pipe 1 ip4 from any to any via vlan12 layer2
65535  19  4943 allow ip from any to any
```

Listing 3: IPFW rules for the topology of Figure 2

3.2.7 Dummynet Pipes

Dummynet pipes are created inside the FreeBSD traffic shaper node by IPFW. They are used for simulating the network adverse conditions such as; delay, bandwidth limitation, probability drop rate, various queueing techniques.

4. PRELIMINARY RESULTS

In this section we provide some of the preliminary results that where captured as part the verification and validation of viNEX. At this stage; it should be emphasized that viNEX is by no means complete, it is in a functional state where basic networking can be accomplished.

Two experiments were run on the six-node dumbbell topology as depicted in Figure 4 below. The main objective of these experiments is to verify if TCP protocol behaves as expected when deployed on viNEX nodes. The first experiment is used to assess the maximum possible bandwidth on viNEX running without traffic loss or delay issues; the second experiment investigates the effects of imposing a delayed and lossy link between Node-3 and Node-4.

4.1 TCP stack and analysis tools used

Since all the nodes envolved in the experiments are NetBSD 4 nodes, we are making use of the latest TCP stack implemented on NetBSD 4, i.e. Reno and NewReno TCP. NewReno TCP is enabled by default in NetBSD. Table 2 lists all the TCP settings that remained constant between Experiment 1 and Experiment 2. NetBSD's NMBCLUSTERS setting was adjusted to 16484 and the kernel was recompiled. In both experiments, we have used the same synthetic load in order make performance comparisons trivial, files of sizes 5MB, 10MB, 20MB, 50MB and 100MB were transferred. In both experiments, data is transferred from Node-1 to Node-6 via the link between Node-3 and Node-4 (see Figure 4). We used the tool `iperf` [14] for traffic generation and maximum bandwidth measurement. TCP flow data packets were captured using `tcpdump` [23] and the tool `tcptrace` [16] was used to analyze them. `xplot` [21] was initially used for graphing but eventually converted xplot datasets to `gnuplot` [9] and used it for graphing. Data conversion was done using the `xpl2gpl` script (located at [16]).

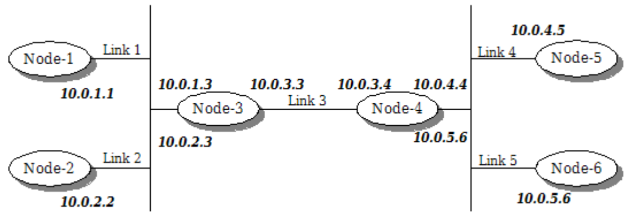

Figure 4: A six-node *dumbbell* topology used for experiments

Table 2: TCP stack settings and configurations

TCP Stack Version	*NewReno + SACK enabled*
Initial Congestion Window	*4 (4068 bytes)*
Send Buffer Maximum	*32Kb*
Receive Buffer Maximum	*64Kb)*
RFC1323 Enhancements	Enabled
NMBCLUSTERS	16384
Traffic Source and Sink	Node-1 and Node-2
Bandwidth Measurement Tool	iperf

4.2 Experiment 1: Maximum bandwidth

The following results were obtained using `iperf` to send data files of sizes 5, 10, 50 and 100MB. TCP statistics were obtained using `tcptrace` tool;

Table 3: Results without any delay and loss

Filesize	5MB	50MB	100MB
Bandwidth (Kb/s)	302.5	320.3	154.43
Data packets:	2405	39400	78483
Ack packets:	1565	25603	51196
Total packets:	3974	65007	129683
Dropped/Rexmt pkts:	0	0	0
Duration (MM:ss.mmm):	0:10.561	02:43.736	11:19.054

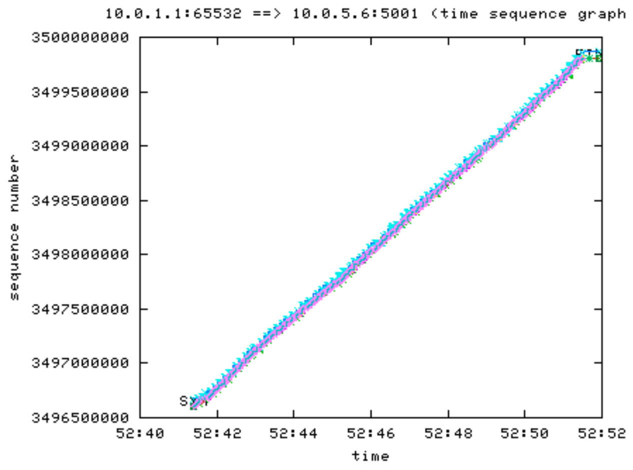

Figure 5: Time Sequence Graph Graph for traffic from Node-1 to Node-2

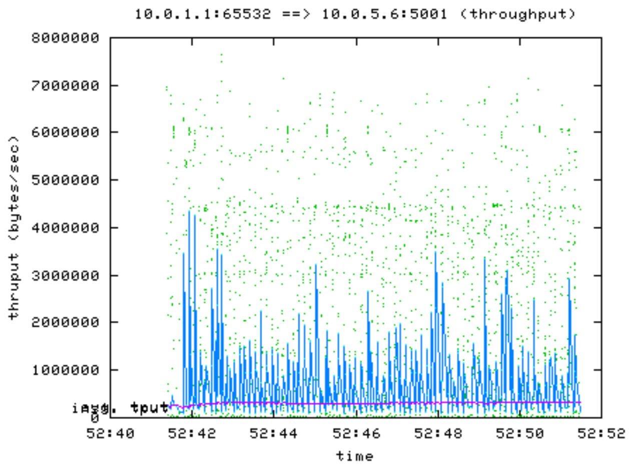

Figure 6: Throughput for the traffic from Node-1 to Node-2

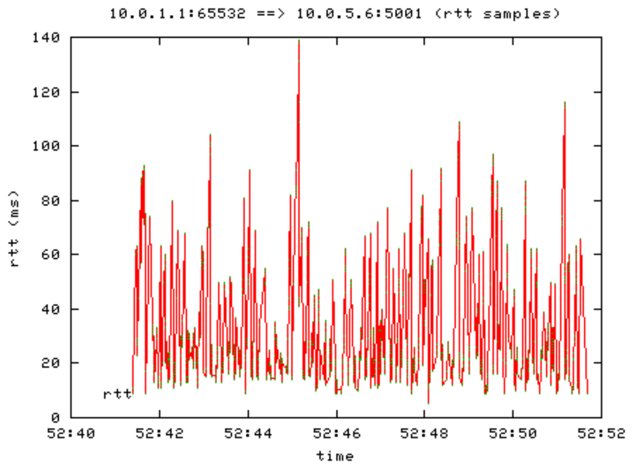

Figure 7: Round Trip Time (RTT) graph for the traffic from Node-1 to Node-2

4.3 Experiment 2: Delay and lossy links

For this experiment, the propagation delay of 20ms and random packet loss of 5% was configured on the link between Node-3 and Node-4. The same data files were transmitted and the results are shown in Table 4 below.

Table 4: Results 20ms delay and 5% loss rate

Filesize	5MB	50MB	100MB
Bandwidth (Kb/s)	72.21	70.67	67.97
Data packets:	4129	41053	82071
Ack packets:	2987	29449	58879
Total packets:	7119	70506	140954
Dropped/Rexmt pkts:	229	2085	4156
Duration (MM:ss.mmm):	01:12.724	12:22.036	25:42.748

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5745
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5745

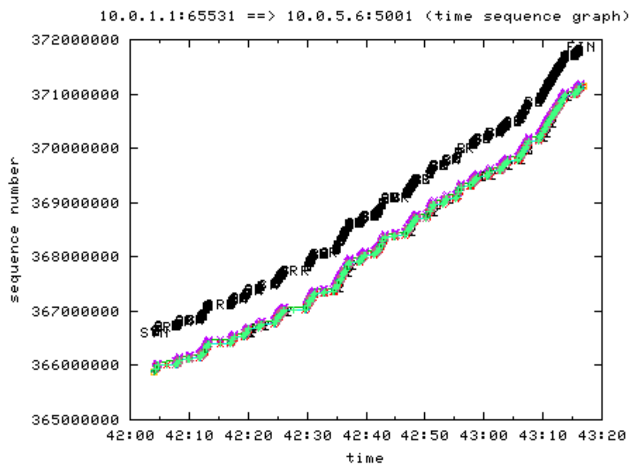

Figure 8: Time Sequence Graph Graph for delayed traffic from Node-1 to Node-2

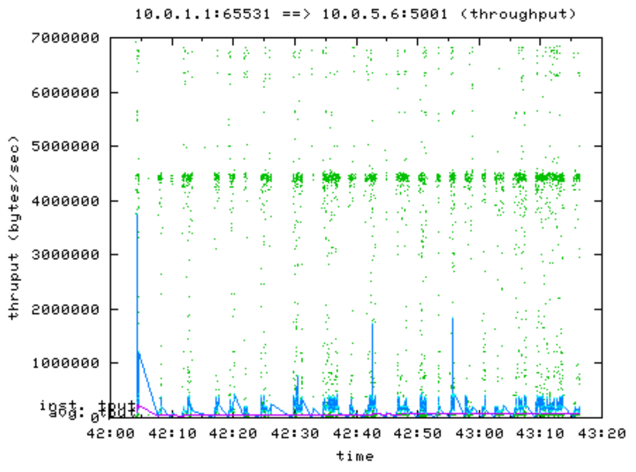

Figure 9: Throughput for delayed traffic from Node-1 to Node-2

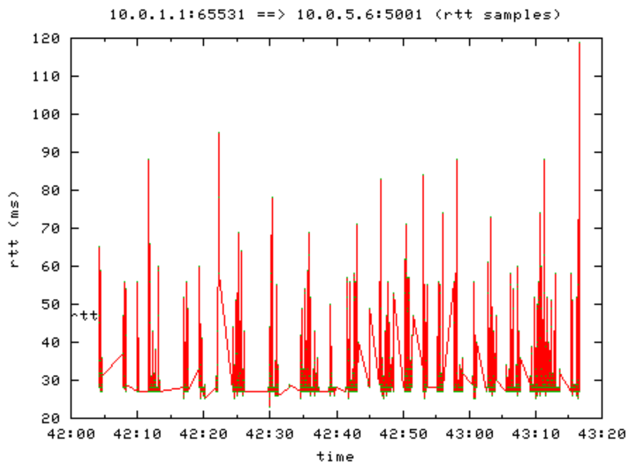

Figure 10: Round Trip Time (RTT) graph for delayed traffic from Node-1 to Node-2

5. RELATED WORK

Virtualization of network emulators is currently receiving a lot of research attention. During the time of this research, we have managed to identify a few number of research work in this space.

The first significant virtualization identified was the current large-scale virtualization initiative being done on Emulab. Instead of using a traditional virtualization tool like Xen, Emulab have chosen the approach of using FreeBSD Jail mechanism. FreeBSD Jail provide a light weight virtualization mechanism through process isolation. See [10] for a detailed description of Emulab's virtualization approach.

UML (User-Mode Linux) has been used quite extensively in virtualizing network emulation, this includes some key research in; 1) the work done using UML (User-Mode Linux) at [22], mainly targeted at evaluating VPN networks, UML was used to evaluate VPN protocols such as PPTP (Point-to-Point-Tunneling-Protocol) and IPSec (IP Security). 2) the UML based emulator for MPLS networks [1], 3) UML was also used in the implementation of VNUML (Virtual Network UML) [8], VNUML is mainly targeted at the evaluation of ipv6 routing protocols.

FreeBSD has also been used to virtualize network emulators; 1) the FreeBSD network stack was virtualized through the cloning technique that allows for multiple network stacks on the same kernel as proposed in [25]. This approach depends on the FreeBSD Jail [18] framework for application environment isolation. Each instance of the protocol stack resembles a full network stack capable of running network routing protocols as well as networking applications. 2) IMUNES is another example, it was proposed in [12]. IMUNES also extends the FreeBSD kernel by enabling it to maintain several networking stacks that are used to run different networking applications. 3) ENTRAPID introduced the approach of virtualizing different 4.4BSD kernels. This enbaled the deployment of different network protocol stacks on the virtualized kernels. ENTRAPID is described in [11].

Further examples of network emulators virtual attempts include; the hypervisor based testbed at [3] aimed at conducting network security experiments, the virtual integrated TCP testbed (or VITT) aimed at evaluating TCP performance at [2], another research is looking at the possibility of using paravirtualization as the basis for a federated PlanetLab architecture at [4] - PlanetLab [17] is a testbed aimed at rapid prototyping and testing of Internet based experiments.

6. CONCLUSIONS AND FUTURE WORK

viNEX is currently a work in progress system in the sense of being able to create nodes, configure links, and route traffic. The reason viNEX was built to investigate the limits to this approach. We are aware about the potential limits of this aproach and we are in the process to establish them.

To take this research work further, three key challenges have been identified. (1) Network performance; we are not impressed by the bandwidth rates obtained in Experiment 1 and 2 above ($\tilde{3}$ Mbit/s) - the slow performance is mainly attributed to the use of QEMU for device emulation. Future enhancements on the XEN HVM are in the pipeline and this limitation might be eliminated. (2) there is a need to identify the class of network experiments that are suitable to be run on viNEX. During the evaluation phase, we were able to deploy the standard IP protocols on viNEX without any issues, e.g. the RIPv2 protocol was deployed on the experiment topology nodes without any modification.

Our emulator (viNEX) was developed using open source technologies; with the major technologies being Xen, FreeBSD, and NetBSD. As a result, we experienced a significant amount of the benefits and advantages of open source, such as; (1) the direct access to the primary software authors and experts, (2) the ability to modify the source to meet our custom requirements, Xen kernel was recompiled with setting `vmxenabled=yes` to enable booting of NetBSD guest kernel, (3) the ability to scale to arbitrary instances without artificial licensing constraints, arbitrary number of NetBSD instances can be booted without any licensing restrictions (4) the ability to learn from the availability of source code (5) the potential to bundle, package and distribute without additional licensing transaction costs.

There is also another opportunity to improve this research by enabling the configuration of the testbed to be done using the NS-2 tool. Key integration points will be identified in order assist interested reader to extend this work. NS-2 is a famous tool in the network research space and therefore is makes sense to use NS-2 as the modeling language for viNEX. Network researchers are already familiar with NS-2 and therefore it will make a seamless adoption of viNEX into their space.

7. REFERENCES

[1] R. Balachander and P. Venkataram. User-mode linux based mpls emulator. *TENCON 2004. 2004 IEEE Region 10 Conference*, B:601–604 Vol. 2, Nov. 2004.

[2] Carlo Caini, Rosario Firrincieli, Renzo Davoli, and Daniele Lacamera. Virtual integrated tcp testbed (vitt). In *TridentCom '08: Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*, pages 1–6, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[3] Dan Duchamp and Greg De Angelis. A hypervisor based security testbed. In *DETER: Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, pages 3–3, Berkeley, CA, USA, 2007. USENIX Association.

[4] Chris Edwards and Aaron Harwood. Using para-virtualization as the basis for a federated planetlab architecture. In *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, page 13, Washington, DC, USA, 2006. IEEE Computer Society.

[5] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, L. Mathy, and T. Schooley. Evaluating xen for router virtualization. *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 1256–1261, Aug. 2007.

[6] Emulab. Emulab home. www.emulab.net.

[7] Kevin Fall. Network emulation in the vint/ns simulator. In *ISCC '99: Proceedings of the The Fourth IEEE Symposium on Computers and Communications*, page 244, Washington, DC, USA, 1999. IEEE Computer Society.

[8] D. Fernandez, T. de Miguel, and F. Galan. Study and emulation of ipv6 internet-exchange-based addressing models. *Communications Magazine, IEEE*, 42(1):105–112, Jan 2004.

[9] gnuplot. Gnuplot. http://www.gnuplot.info/.

[10] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 113–128, Berkeley, CA, USA, 2008. USENIX Association.

[11] X.W. Huang, R. Sharma, and S. Keshav. The entrapid protocol development environment. *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 3:1107–1115 vol.3, Mar 1999.

[12] Miljenko Mikuc Marko Zec. Operating system support for integrated network emulation in imunes. In *In Proc. of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS), Boston, MA, 2004.*, 2004.

[13] Mukosi Abraham Mukwevho. vinex home. http://sites.google.com/site/mukosi/.

[14] NLANR/DAST. Iperf. http://sourceforge.net/projects/iperf.

[15] NS-2. Ns-2 wiki. http://nsnam.isi.edu/nsnam/index.php/.

[16] Shawn Ostermann. tcptrace. http://www.tcptrace.org/.

[17] PlanetLab. Planetlab. http://www.planet-lab.org.

[18] Robert N. M. Watson Poul-Henning Kamp. Jails: Confining the omnipotent root. In *2nd SANE Conference*, May 2000.

[19] Ian Pratt, Keir Fraser, Steven Hand, Christian Limpach, Andrew Warfield, Dan Magenheimer, Jun Nakajima, and Asit Mallick. Xen 3.0 and the art of virtualization. In *Proceedings of Linux Symposium 2005*, July 2005.

[20] Paul 'Rusty' Russell. Ebtables firewalling. http://ebtables.sourceforge.net/.

[21] Timothy Jason Shepard. xplot. www.xplot.org.

[22] Ralf Spenneberg. Emulating networks using user-mode linux. http://www.samag.com/documents/s=8997/sam0401a/0401a.ht

[23] Craig Leres Van Jacobson and Steven McCanne. tcpdump/libpcap. http://www.tcpdump.org/.

[24] Xen. Xen home. http://www.xen.org/.

[25] Marko Zec. Implementing a clonable network stack in the freebsd kernel. In *In Proceedings of the USENIX 2003 Annual Technical Conference*, pages 137–150, 2003.