Technical Report WINLAB-TR-331*[1]

Evaluation Of OpenVZ Based Wireless Testbed Virtualization

Gautam Bhanage, Ivan Seskar,
Yanyong Zhang, Dipankar Raychaudhuri
Rutgers University

August, 2008

RUTGERS WIRELESS INFORMATION

NETWORK LABORATORY

Rutgers - The State University of New Jersey

Technology Center Of New Jersey

671, Route 1 South

North Brunswick, New Jersey 08902-3390

Phone: (732) 932-6857 FAX: (732) 932-6882

[1]*The authors may be contacted by e-mail at gautamb, seskar, yyzhang, ray@winlab.rutgers.edu.

Technical Report WINLAB-TR-331[*2]

Evaluation Of OpenVZ Based Wireless Testbed Virtualization

Gautam Bhanage, Ivan Seskar,

Yanyong Zhang, Dipankar Raychaudhuri

Evaluation Of OpenVZ Based Wireless Testbed Virtualization

Gautam Bhanage, Ivan Seskar,

Yanyong Zhang, Dipankar Raychaudhuri

WIRELESS INFORMATION NETWORK LABORATORY

Rutgers - The State University of New Jersey

Technology Center Of New Jersey

671, Route 1 South

North Brunswick, New Jersey 08902-3390

Phone: 732-932-6857 FAX: 732-932-6882

ABSTRACT

Wireless testbed virtualization aims to provide a stable, reproducible, environment that is capable of simultaneously supporting multiple radio experiments in the same RF environment. Our study provides a brief overview of current effort in the direction of virtualization, and aspects affecting the feasibility of such setups. Our preliminary studies involve comparison of current virtualization schemes and their suitability for wireless testbed virtualization.

Table of Contents

List of Abbreviations

MAC - Medium access control

VM - Virtual Machine

VPS - Virtual Private Servers

VETH - Virtual ethernet device

VENET - Virtual network device

UML - User Mode Linux

RTT - Round trip time

FDM - Frequency division multiplexing

TDM - Time division multiplexing

SDM - Space division multiplexing

TCP - Transmission control protocol

FTP - File transfer protocol

RTP - Real time protocol

1.1 Introduction

Virtualization of wireless network resources allows provision of capabilities for handling multiple concurrent experiments ("slices") on the same set of radio devices. Strict virtualization of wireless network elements is fundamentally a difficult problem because of the fact that radios interfere with other radios in their immediate vicinity via interactions at RF spectrum, physical (PHY) and medium-access control (MAC) layers. These interactions make it more difficult to create orthogonal time slices on radio channels than on wired network links. Virtualization of testbed resources in ORBIT [2] is motivated by two factors:

- GENI Integration: GENI [3] proof-of-concept work started in 2006-07 required the capability of integrating virtualized PlanetLab and VINI networks with ORBIT [10]. GENI also requires integration of control and management across wired and wireless networks, providing researchers with a single programming interface and experimental methodology. An integrated prototype is extremely valuable in building a practical understanding of integration issues of wired and wireless networks.
- Testbed Utilization: ORBIT [2] is a two-tier laboratory emulator/field trial network testbed which typically uses a time shared experimentation model. Each experimenter can reserve the grid for approximately two hours and has complete control of the grid. Heavy use of the testbed dictates a need to increase experiment temporal throughput by sharing grid resources where possible. Wireless virtualization aims to support concurrent experiments on shared wireless network resources.

Eventual goals of this project are:

1. Identification of a node virtualization scheme based on existing solutions that best fit testbed virtualization.
2. Design and implementation of a mechanism that allows flexible sharing of radio resources.
3. Design and implementation of a job scheduler that selects, maps and monitors compatible wireless experiments.

Through this study we aim to achieve the first step in the process by determining criteria, and performing comparison of schemes to select an approach suitable for wireless testbed virtualization.

Section 1.2 will give a brief overview of virtualization techniques and current testbed architectures using these techniques. Sections 1.3 and 1.5 provide a qualitative and quantitative

comparison of some virtualization schemes we considered for our setup. Finally, Section 1.6 provides a few conclusions and talks about future directions.

Terminology: From here on, the *host* operating system will refer to the virtualization environment and the *guest* will refer to the instance of the virtualized OS.

1.2 Related

Virtualization techniques are used to share testbed (system) resources. Testbeds like EMULAB [14] and Planetlab [8] provide environments for large scale systems emulation. Planetlab infrastructure consists of a planetary scale service running across a set of distributed nodes. An experimenter can reserve a slice across multiple machines and run experiments across all of the nodes over a period of time. These experiments can be either short term or long term. The needs of our architecture are significantly different from the Planetlab usage model. Our wireless emulation facility requires that the users be able to achieve a deterministic performance from the network rather than a best effort link. Our setup should also allow the user to make restricted privileged calls. E.g. Changing the control parameters of an interface (like power or transmission rate) at run time without allowing multiple users to conflict with each other. Experiences on setup of a large scale wired distributed virtualization schemes are also discussed in [6]. The ModelNet [1] emulator achieves similar large scale flexibility by sacrificing on flexibility by abstractions. Diecast [4] achieves flexible sharing of resources in emulation by slowing down time, thereby providing the illusion that each virtualized instance has access to complete machine resources.

Specific to wireless virtualization we observe two studies. One of them [9] compares the performance of virtualizing the channel in space versus in time while the other [11] focusses on virtualization in frequency with the use of UML. Through our study we will validate and compare performance of UML with that of OpenVZ and attempt to quantify difference in performance with the two schemes.

1.3 Qualitative Comparison

A virtualized system generally has a slightly degraded performance as compared to a non-virtualized one due to sharing of resources. However, when we are considering testbed virtualization we would want these effects to be minimal to achieve reproducible results. The goal of this section is to present a brief comparison of some standard virtualization architecture and their suitability to our application.

Feature	*FullVirtualization*	*ParaVirtualization*	*OSVirtualization*
Intrusiveness	*Least*	*Intermediate*	*Most*
Hypervisor	*Yes*	*Yes*	*No*
SystemCallPerformance	*Degrades*	*intermediate*	*Best*
Kernel	*Individual*	*Individual*	*Shared*

Figure 1.1: A comparison of techniques for system virtualization. It is important to note here that full virtualization refers to systems running without native virtualization support. Intrusiveness refers to the level of changes that need to be made to the guest operating system. System call performance refers to the efficiency with which privileged calls are executed with each of the architectures.

Production scale virtualization systems can be broadly classified as shown in Figure 1.Full virtualization [13] refers to direct hardware emulation which uses a hypervisor to trap and execute privileged calls on the fly[1]. One of the main advantages of full virtualization is it is least intrusive and allows running of unmodified operating systems. Paravirtualization [7] is similar to full virtualization and uses a hypervisor to trap and execute system calls. However, para-virtualization requires changes to the guest operating system which allow the virtual machine monitor to work along with the hypervisor in reducing penalties associated with system calls. The most intrusive form of virtualization is operating system based where the virtualized systems run as processes in the operating system. The host operating system is modified to provide secure isolation of the guest operating systems also referred to as virtual machines (VM) or virtual private servers (VPS). Since our design relies on controlling most of the frame-work with a limited variety of guest operating systems we are considering two prominent schemes for OS virtualization:

- OpenVZ [12]
- User Mode Linux (UML) [5]

A follow up of this study will include performance comparison with Xen.

Some of the desired features of the virtualization scheme for suitable working with our proposed testbed architecture are:

1. Controlling disk quotas
2. Enabling control of CPU utilization per sliver for ensuring experiment repeatability.
3. Some control over other system features such as memory, TCP sockets, and IP packets.

[1]Native virtualization refers to the systems where the processor has support for virtualization and allows multiple unmodified operating systems to run together. Full virtualization does not refer to these systems.

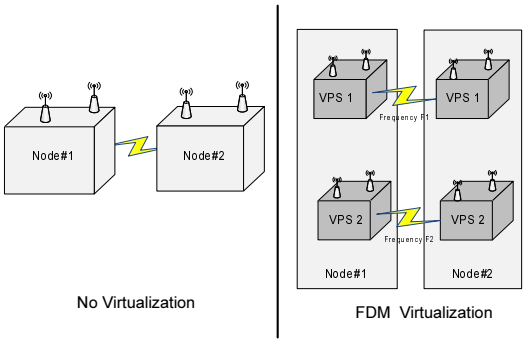


Figure 1.2: Experiment setup for OpenVZ evaluation

4. Flexible mapping of standard Linux distributions.

5. Ease of wireless driver integration with provision for flexible on the fly reconfiguration.

Each of these systems provide hooks for flexible design and a lot of the decision depends on the empirical performance of these systems.

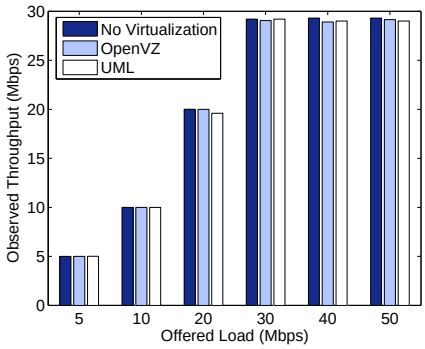
Terminology: *Throughout the rest of this document guest will refer to the virtualized operating system and the host will refer to the system running the virtualized instances.*

1.4 Experimental setup and Methodology

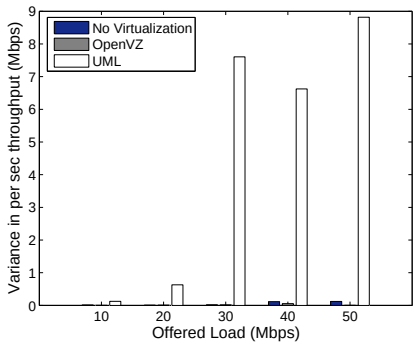
This section provides a brief overview of the ORBIT testbed where the experiments were conducted, introduction to OpenVZ and finally the experiment setup used for evaluation.

Parameter	*Value*
Channel	36
Tansmit Rate	36Mbps
Offered Load	Time Varying
Experiment Duration	3 Minutes
Averaging Duration	Per Second
Operation Mode	802.11a
Chipset	Atheros
Driver	MadWiFi 0.9.3

Figure 1.3: Experimental Parameters Used With ORBIT Nodes. The Madwifi drivers are compiled with transmitter diversity disabled.



(a) Average UDP throughput comparison.



(b) Variance in UDP bandwidth.

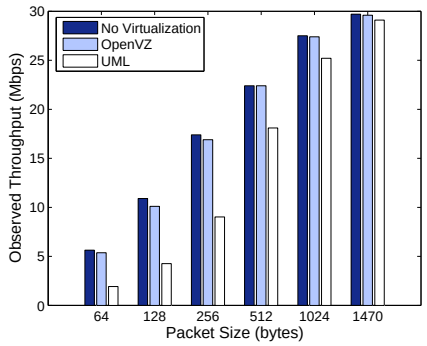
Figure 1.4: UDP throughput and variance in throughput as measured with different schemes. Performance is measured as a function of offered load per flow with a fixed packet size of 1024bytes. Variance in UDP bandwidth is measured over per second observed throughput at the receiver.

1.4.1 Testbed Summary

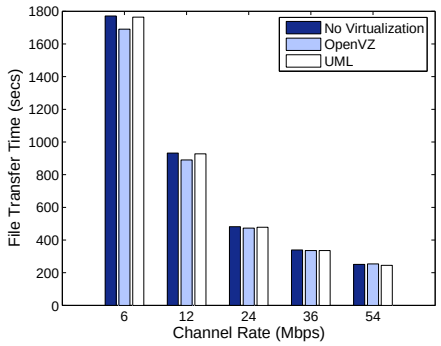
ORBIT is a two-tier laboratory emulator/field trial network testbed designed to achieve reproducibility of experimentation, while also supporting evaluation of protocols and applications in real-world settings. The setup consists of a large two dimensional grid of 400 802.11 radio nodes with reproducible wireless channel models. Every node is a small form factor PC with 1GHz Via C3 CPU, 512 MB RAM, 20 GB hard disk and three ethernet ports, one of which is used for node configuration and control. Majority of the ORBIT nodes are fitted with Atheros 5212 based IEEE 802.11 a/b/g cards while remaining have Intel cards. We used Atheros cards for our experimentation.

1.4.2 OpenVZ Overview

OpenVZ allows loading of multiple standard distributions through the *template* mechanism. Each instance of the guest operating system is referred to as a virtual private server (VPS). Other features such as disk quotas, memory, and cpu units can be controlled through the resource control functionality built in to the OpenVZ system. Two types of network devices are supported. A virtual ethernet device (vethX) is an ethernet like device that can be used inside a VPS. The veth device can be assigned an IP address and can be bridged with ethX. A venet device is a point to point link between the container and the host system and works at the IP-layer. A venet device is created automatically and can be assigned IP addresses using the vzctl commands. For convenience and ease of use we employ the venet interfaces.



(a) Difference in UDP throughput with varying packet sizes.



(b) FTP performance with 1GB file transfer.

Figure 1.5: Measurement of UDP throughput with varying packet sizes and file transfer time with FTP. For the UDP throughput measurement, channel rate is constant at 36Mbps and packet size is varied. For the FTP experiment, packet size is constant at 1024 and channel rate is varied.

1.4.3 Setup

Figure 1.2 shows the experiment setup used for determining the performance of the system. Each machine runs two VPSs. Each of the VPSs are mapped to two independent wireless interfaces operating on orthogonal channels. *venet0* devices from each VPSs are configured to map to corresponding hardware devices on the host. In the non- virtualized case, we run a single wireless link between two nodes. With the virtualized case we wish to test the degradation in performance by running two identical links between the same nodes on orthogonal wireless channels. For sake of evaluation we compare some of our results from OpenVZ with those obtained by running UML. Generic experiment parameters are as shown in the Figure 1.3.

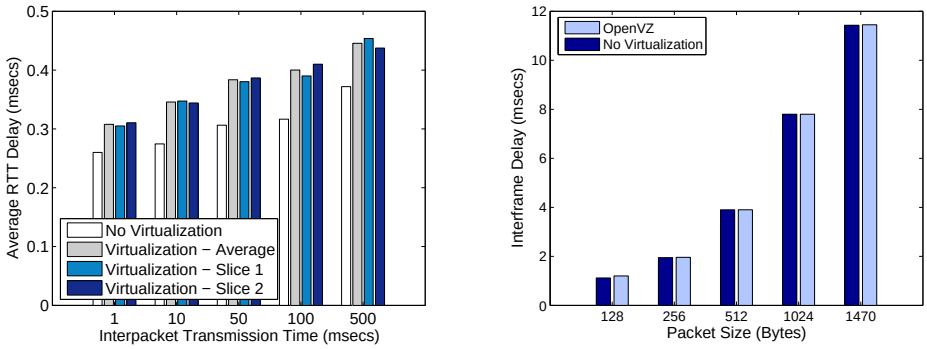
1.5 Quantitative Evaluation

We start the evaluation by measuring the overheads in throughput and delay, followed by measurement of isolation achievable between slices.

1.5.1 Overhead Evaluation

Overhead evaluation involves measurement of system performance with virtualization as compared to that without virtualization. To measure performance with virtualization we use the OpenVZ (and UML for throughput comparison) as a part of the FDM virtualization setup.

The goal of the first experiment is to compare the saturation UDP bandwidth achievable with both the setups as shown in Figure 1.2. Results obtained from these experiments are



(a) Average RTT delay.

(b) Interframe Space Vs Packet Sizes.

Figure 1.6: Delay as measured for different experiment scenarios. Minimum and average round trip time measurements are based on ping while interframe space measurements are based on difference in time stamps between packets arriving at the receiver.

as shown in Figure 1.4(a). The figure plots the observed UDP bandwidth for different link offered loads. Throughput obtained in the virtualized case are averaged over the two links. It can be observed that both below and above channel saturation there is no distinct difference in throughput with either OpenVZ or UML as compared to the case with no virtualization. This tends to indicate that both systems perform efficiently (with least overhead) under tested conditions.

Since the results shown in the previous experiments were averaged over a period of 180 seconds, we wished to evaluate the variance in throughput with each of the schemes. Figure 1.4(b) plots the per second variance in throughput with OpenVZ, UML and no virtualization for different offered loads. It can be observed that below saturation, the variance is low for all the schemes. However, close to and above saturation, it can be observed that the variance with UML increases. Superior performance of OpenVZ may be associated with tighter scheduling and better isolation.

To test the performance with varying packet sizes we evaluated the net throughput observed with the three schemes for a constant offered load and transmission rate. Figure 1.5(a) shows the performance with varying packet sizes at a constant offered load of 40Mbps and constant channel rate of 36Mbps. The results show that for the maximum packet size of 1470 bytes, the performance with the virtualized cases of OpenVZ and UML are comparable to that without virtualization. However, for all sizes of 1024bytes and below, we see that UML has a significantly higher packet processing overhead which leads to a degraded performance. Degradation in performance with UML may be attributed to the lack of support for virtualization in the host kernel leading to higher overheads.

To measure the FTP performance of the system we setup a file transfer of 1GB. Results presented in Figure 1.5(b) measure the the time taken for the file transfer with varying channel rates. We observe that for all the considered channel rates there is very little performance difference with the two virtualization schemes.

1.5.2 Overhead Evaluation - Delay

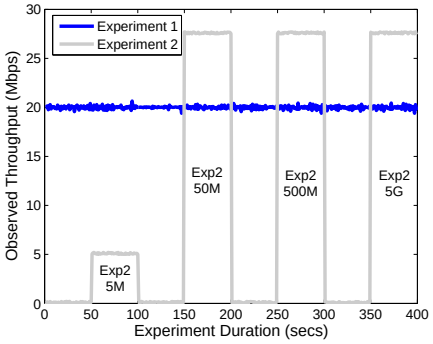
Delay measurements are made with the same setup as that used for the throughput experiments. These are typically important for experiments that measure performance of real time systems or data. Delay measurements were setup with two goals:

- Distribution of delay across slices.
- Distribution of delay overhead with packet sizes.

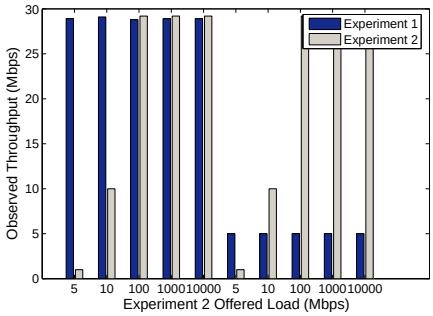
To measure the distribution of delay across slices we run ICMP traffic (ping) across both slices. All measured delay values are round trip times (RTT). Figure 1.6(a) measures the delay performance with varying inter-packet durations. Inter-packet transmission times, is the time duration between two consecutive packets as seen at the sender. For each sending rate, readings are averaged over an interval of 300 secs. Delay values are plotted for the case with no virtualization, average delay across both the slices, and delay across individual slices. The results show that in all cases, virtualization adds a very small average overhead (of the order of $0.05msec$) in terms of absolute delay. The RTT delays for slices increase slightly with lesser sending rates due to slight decrease in CPU time spent on network tasks. Despite the overhead being negligible, we notice that the performance across both slices is always comparable. Thus we see that OpenVZ adds little or no overhead to the delay measurements and it is safe for making temporal measurements across slices.

Theoretically we would expect delays to increase with packet sizes, due to higher processing delays. The goal of these measurements is to verify if using OpenVZ results in significantly different performance across different packet sizes. Delay measures the difference in absolute time between when a frame is transmitted and the time when it is received. However, we measure interframe arrival times since one-way delay measurements require explicit synchronization[2]. Let us assume that the delay for two consecutive frames i and j is given as D_i and D_j. Each D_k is evaluated as the difference in the time stamp at the receiver (R_k) from the sender S_k.

[2]Synchronization across machines is possible to a granularity of a few msecs using the network timing protocol daemon. However, achieving higher granularity is difficult using standard tools. Delay measurements can easily replace the inter-frame arrival measurements if this issue is fixed.



(a) UDP transient performance.



(b) Cross coupling between experiments.

Figure 1.7: Experiments for measuring the cross coupling and interference between experiments. First plot shows a performance with time, while the second plot displays results averaged over 180secs.

We can evaluate the difference in delay of two consecutive frames i and j, given by $D_j - D_i$ as:

$$(R_j - S_j) - (R_i - S_i) = (R_j - R_i) - (S_j - S_i) \tag{1.1}$$

Hence the difference in inter-frame arrival times given by $R_{diff} = R_j - R_i$ is evaluated as:

$$R_{diff} = (D_j - D_i) + (S_j - S_i) = D_{diff} + \delta \tag{1.2}$$

where D_{diff} represents difference in delay of consecutive frames and δ denotes difference in time stamps of consecutive frames at the sender. As long as the frames are sent at a constant rate over a single link, with static channel conditions, δ appears as a constant value. Thus we can use difference in inter-frame arrival times as an approximation for difference in delays of consecutive frames.

Measurements in Figure 1.6(b) are based on the average difference in receiver time stamps for *ten thousand* consecutive frames. The results show that increase in packet sizes results in increasing inter-frame delays, thus indicating increasing packet delays. Increasing the packet sizes results in little or no difference between the measurements with virtualization and without. This confirms the notion that OpenVZ adds little overhead in delay measurements, and the overhead is limited across all packet sizes.

1.5.3 Slice Isolation

An important requirement of a virtualized testbed is that if the results are repeatable in the non-virtualized case, they should be repeatable in the virtualized environment. With such a requirement, experiment repeatability in the virtualized case is going to be directly proportional

to the isolation between experiments. Since OpenVZ has clearly outperformed UML in the previous experiments we will determine the performance isolation provided by OpenVZ in this section. We will first evaluate the transient response of the system followed by measurement of experiment cross coupling across slices.

In our experiment for measuring transient response in throughout of one experiment in response to the change in throughput of the other, we maintained the offered load for the experiment running on slice 1 at a constant value of 20Mbps. Figure 1.7(a) shows the performance of that experiment when the offered load of the experiment on slice 2 is varied as a function of time. The offered load of experiment 2 is varied as 5M, 50M, 500M, and 1000M. We can see that there is little or no correlation in the throughput of the experiment running on slice 1 in response to the experiment running on slice 2. Hence OpenVZ provides excellent isolation in terms of the observed transient response of the experiments.

We define experiment cross coupling as the difference in throughput with virtualization as a percentage of the throughput without virtualization. To measure cross coupling we maintain the offered load of experiment in slice 1 at constant values of 30Mbps initially followed by 5Mbps for the second experiment. In the mean time we vary the offered load of the experiment on slice 2, as 5M, 10M, 100M, 1G, 10G. Results of the experiments shown in Figure 1.7(b) show that there is negligible cross coupling, and the results of the experiments in slice 1 are never affected. It is important to note that these results are achieved without tweaking features of OpenVZ that allow setting of custom cpu usage per slice.

1.6 Conclusion and Future work

Preliminary investigations of the setup with OpenVZ reveal a superior performance as compared to that with UML. Betterment in results may be attributed to a tighter virtualizing mechanism and a more effective approach for network setup. Future work involves:

1. A comparison involving a para-virtualization scheme like Xen.

2. Integration of selected mechanism in an automated virtualization framework.

References

[1] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *In Proc. of the 5th Symp. on Operating Systems Design and Impl. (OSDI)*, dec 2002.

[2] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. In *IEEE WCNC*, March 2005.

[3] GENI Design Principles. Global Environment For Network Innovations (GENI). In .

[4] D. Gupta, K. V. Vishwanath, and A. Vahdat. Diecast: Testing distributed systems with an accurate scale model. In *Proceedings of the 5th ACM/USENIX Symposium on Networked Systems Design and Implementation*. USENIX, April 2008.

[5] Jeff Dike. A user-mode port of the Linux kernel. In *5th Annual Linux Showcase and Conference, Oakland, California*, Nov 2001.

[6] Mike Hibler and Robert Ricci and Leigh Stoller and Jonathon Duerig and Shashi Guruprasad and Tim Stacky and Kirk Webby and Jay Lepreau. Large-scale Virtualization in the Emulab Network Testbed. In *USENIX*, 2008.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warbeld. Xen and the Art of Virtualization. In *In Proc. of the 19th ACM Symp. on Operating Systems Principles (SOSP)*, oct 2003.

[8] L. Peterson, S. Muir, T. Roscoe, and A. Klingaman. PlanetLab Architecture: An Overview. Technical Report PDN–06–031, PlanetLab Consortium, May 2006.

[9] R. Mahindra, G. Bhanage, G. Hadjichristofi, I. Seskar, D. Raychaudhuri, and Y. Zhang. Space Versus Time Separation for wireless virtualization On an Indoor Grid. In *Next Generation Internet (NGI) testbeds*, Mar 2008.

[10] S. Paul and S. Seshan. Virtualization and Slicing of Wireless Networks. In *Technical Report GENI Design Document 06-17, GENI Wireless Working Group*, sep 2006.

[11] S. Singhal, G. Hadjichristofi, I. Seskar, and D. Raychaudhuri. Evaluation of UML based wireless network virtualization. In *Next Generation Internet (NGI) testbeds*, Mar 2008.

[12] SWSoft manual. OpenVZ Instruction Manual. In .

[13] VMWare player for virtualization. .

[14] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002. USENIX Association.