Experimental Evaluation Of OpenVZ From A Testbed Deployment Perspective [*]

Gautam Bhanage, Ivan Seskar, Yanyong Zhang, Dipankar Raychaudhuri, and Shweta Jain

WINLAB, Rutgers University, North Brunswick 08902, USA
{gautamb, seskar, yyzhang, ray, sjain}@winlab.rutgers.edu,
WWW home page: http://www.winlab.rutgers.edu

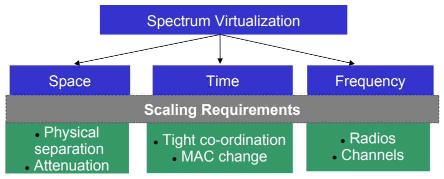
Abstract. A scalable approach to building large scale experimentation testbeds involves multiplexing the system resources for better utilization. Virtualization provides a convenient means of sharing testbed resources among experimenters. The degree of programmability and isolation achieved with such a setup is largely dependent on the type of technology used for virtualization. We consider OpenVZ and User Mode Linux (UML) for virtualization of the ORBIT wireless testbed and evaluate their relative merit. Our results show that OpenVZ, an operating system level virtualization mechanism significantly outperforms UML in terms of system overheads and performance isolation. We discuss both qualitative and quantitative performance features which could serve as guidelines for selection of a virtualization scheme for similar testbeds.

1 Introduction

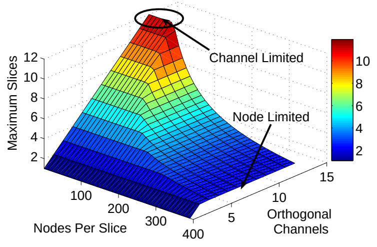
Experimental validation of research ideas in a realistic environment forms an important step in identifying many practical problems. This is specially true for wireless networks since wireless communication environment is hard to accurately model through simulations. Public access testbeds like ORBIT [12, 17, 23], provide the research community with platforms to conduct experiments. ORBIT [12], typically uses a time shared experimentation model where each experimenter can reserve the grid nodes for a fixed duration (slot - approximately two hours) and has complete control of these nodes during the reservation period. An ever increasing demand for grid slots can only be met through sharing of the testbed whenever possible. Since spatial expansion is not an economically viable solution due to the limited space, prohibitive cost of setup and maintenance, we propose virtualization of ORBIT to support simultaneous experiments. Wired testbeds like VINI [9] and Planet lab already use node and network virtualization for the same reason. In our study, we will cater specifically to requirements for sharing a *wireless* testbed through virtualization.

Another important motivation for ORBIT testbed virtualization is the integration with the GENI [1] framework. This requires ORBIT to be virtualized for

[*] Research supported in part by the National Science Foundation grant#CNS-072505



(a) Spectrum sharing options.



(b) FDM virtualization scaling.

Fig. 1. Options for sharing radio resources on ORBIT and potential capacity of the ORBIT grid with 800 interfaces(2/node), 12 channels(802.11a), 2VMs/Node

allowing integration with other shared testbeds such as PlanetLab and VINI [19, 7]. GENI also requires combining of control and management across wired and wireless networks, providing researchers with a single programming interface and experimental methodology. Since the ORBIT testbed currently supports only a single experimenter mode of operation, virtualization is essential for integration.

While wired network virtualization may be achieved by pre-allocating memory, CPU cycles and network bandwidth, to achieve perfect virtualization of the wireless network, we need to perfectly isolate both the physical devices and the wireless spectrum while providing flexibility for experimentation. This additional requirement makes the problem of wireless virtualization much harder compared to the wired counterpart [21]. Figure 1(a) shows different options for sharing the radio spectrum. The authors in [21] attempt to solve the spectrum sharing problem by separating experiments in time. As observed, time sharing of a single channel can result in a less repeatable performance due to context switching overheads even though it could possibly reduce the wait time for experiments. Due to the availability of a large number of radio interfaces (800 - 2/node), we share the spectrum by allocating orthogonal channels to slices. ORBIT nodes are equipped with two wireless interfaces each and therefore two virtual machines may be run on each node thereby doubling the capacity of the grid. Figure 1(b) shows the potential capacity of the ORBIT grid with such a frequency division (FDM) based virtualization. We observe that the number of simultaneous experiments supported on the grid are limited either by the number of orthogonal channels [1] or the number of nodes allocated per experiment.

In order to provide meaningful experimentation in the virtualized wireless testbed, the choice of the vitualization platform is critical. In this work, we start by identifying the requirements and qualitative issues to consider when selecting a virtualization platform in Section 2. After discussing the relative merits of OpenVZ for our application, we present a comparative experimental evaluation

[1] Experiments that use other wireless technologies like zigbee and GNU radio may be run simultaneously provided they use non-interfering frequencies

Table 1. Comparison of schemes from an ORBIT user perspective

Feature/Experiments	Full - Virtualization	Para - Virtualization	OS - Virtualization
Security Experiments	Yes	Yes	Yes
Network Coding	In Kernel	Overlay*	Overlay*
Mobility and Routing	Yes	Yes	Yes
Rate And Power Control	In Driver	Radiotap**	Radiotap**
Wireless Applications	Yes	Yes	Yes
Phy Measurements	Yes	Yes	Yes
MAC Parameter Control	Yes	Yes	Yes***
Transport layer Modification	In Kernel	*Emulation*$^{\nabla}$	*Emulation*$^{\nabla}$

* Transport layer experiments can be implemented as a part of overlays.

** Radiotap headers allow for per frame rate and power control.

*** For Atheros devices MAC parameters (txop, CW, AIFS)are supported per interface.

∇ Use a click like mechanism on top of IP for custom flow or error control

of UML and OpenVZ in Section 4. Related work is discussed in Section 5. Finally, conclusions and future directions are presented in Section 6.

2 Background and Platform Selection

Production scale virtualization systems can be broadly classified as full, para and OS virtualization. Full virtualization [8, 2](e.g.,VMWare, KVM) refers to a technique that emulates the underlying hardware and uses a software layer called hypervisor that runs directly on top of the host hardware to trap and execute privileged instructions on the fly[2]. Full virtualization is the least intrusive[3] form of system virtualization. In para virtualization [16, 6](e.g., Xen, UML) the hypervisor layer exists within the host operating system to intercept and execute privileged instructions. Unlike full virtualization, para virtualization requires changes to the guest operating system. The most intrusive form of virtualization is operating system based [4](e.g.,OpenVZ) where the virtualized systems run as isolated processes in the host operating system. The host OS is modified to provide secure isolation of the guest OS. For the purpose of this study we lay out the main qualitative criteria and select candidates for performance evaluation based on their suitability for the ORBIT testbed.

Qualitative features of a virtualization scheme which are important from a wireless testbed administrator's perspective are as follows:

1. Ease of administration: Clean API to schedule node resources such as CPU, disk and memory on a per slice basis should be possible.

[2] Native virtualization is a virtualization approach where the processor has support for virtualization e.g.,IBM System/370 and allows multiple unmodified operating systems to run together. Full virtualization does not include these systems.

[3] Intrusiveness refers to the degree of changes that need to be made to the guest OS to get it working with virtualization.

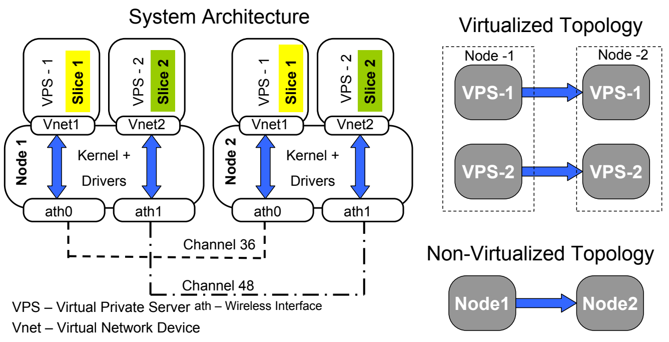


Fig. 2. Experiment setup for OpenVZ evaluation

2. Shared or exclusive interface mapping: The setup should allow flexible mapping of virtual interfaces within the slice to physical interfaces or one or more virtual interfaces (on the hardware like virtual access points).
3. Control over network connectivity: Mechanisms should be available to bandwidth limit slices and control interaction between slices.

All types of virtualization schemes allow for such functions. However, in our experience the most flexible and easy approach for controlling the VMs is through operating system level virtualization such as OpenVZ. Such a setup also allows for the reuse and extension of regular system administration tools (such as IPTABLES, DHCP, SSH, LDAP) for controlling VMs.

From the perspective of an ORBIT experimenter we consider the following:

1. Support for standard and custom Linux distributions: Orbit nodes supports a wide variety of Linux distributions and users are free to use their own customized version. The virtualization platform running on ORBIT must support similar flexibility for the experimenter.
2. Root access within container: This feature is useful for an experimenter for providing complete freedom within the container.

Multiple Linux distributions with root access in VMs are inherently supported in all three forms of virtualization. A more detailed comparison is shown in the Table 1. It is observed that all wireless experiments scenarios can be either directly supported or emulated (using open source radiotap libraries and overlays) with all the virtualization setups. Traffic control elements such as Click [13] can also be run on hosts to allow for bandwidth shaping and interface mapping. Appropriate API can also be exposed from the driver to allow experimenters to have a controlled interaction with the driver. The only experiments not supported in operating-system level virtualization is the option of customizing the host kernel itself to cater to individual VMs. Despite needing emulation to support experiments that would conventionally be done by direct changes in the host kernel, the possibility of obtaining very tight slice isolation [22] make OS-level virtualization a strong candidate for evaluation.

Based on these inferences, the choice of a virtualization mechanism for ORBIT is not limited to any one type. However, full virtualization such as KVM requires specific CPU virtualization extensions (E.g. Intel VT or AMD-V) which are currently not available with our ORBIT boxes, and hence is not considered for evaluation. We consider OpenVZ (OS level) and User Mode Linux (Para - level) virtualization for quantitative comparison with testbed deployment. Since UML based virtualization has been performed in a previous study [20], this study focusses on the performance analysis of OpenVZ. We ruled out Xen in this performance study due to incompatibility with the Via C3 processors used in the ORBIT testbed.

3 Experiment Setup

The Orbit testbed is a two-dimensional grid of 400 small form factor PCs with 1GHz Via C3 CPU, 512 MB RAM, 20 GB hard disk, three ethernet ports (control, data and chasis management) and two WiFi interfaces[4]. We used Atheros 5212 chipset cards with the MadWiFi(0.9.4) [3] drivers for our experiments.

Figure 2 shows our experiment setup. OpenVZ uses the concept of a container also called virtual private server (VPS), an entity that performs like a stand alone server. It also provides a virtual network device named as *venetX* per VPS that acts as a point to point link between the container and the host system. We configure the *venet* devices from each of the two VPSs (on every node) to map to a corresponding WiFi card on the host. Effective virtualized and non-virtualized links are as shown in the figure. The UML virtualization setup is described in detail in [20] and is quite identical to the OpenVZ setup.

We run each experiment for 3 minutes using UML and OpenVZ setups as well as with no virtualization. The operating mode of the WiFi cards was 802.11a with bit rate of 36Mbps set at channel 36. The debian linux distribution (*Woody*) was used for both guest and host operating systems.

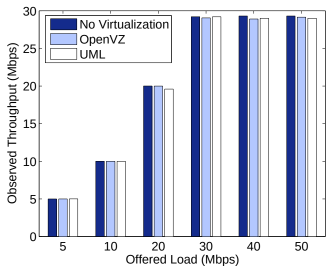
4 Performance Evaluation

We measure overheads in throughput and delay, followed by measurement of *slice isolation* achievable between slices. Quantitative evaluation presented in this section takes into account the importance of different measurement criterion. For instance, the isolation achieved between slices is far more important than sustainable peak throughput as it directly determines experiment repeatability.

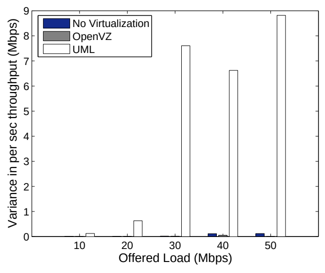
4.1 Throughput Measurements

We use the *iperf* [5] tool to generate saturation UDP traffic and average the througphut over 3 min intervals. We plot the observed UDP throughput with

[4] It should be noted that though the results presented in the following section are hardware specific, performance trends will hold and scale with hardware capacity and load on the system.

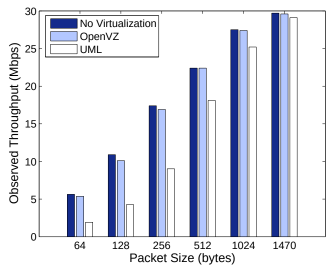


(a) Average UDP throughput comparison.

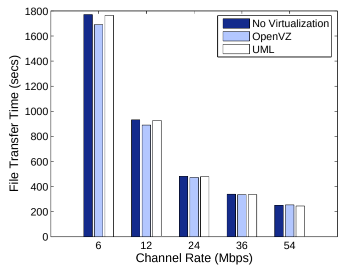


(b) Variance in UDP bandwidth.

Fig. 3. 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.



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

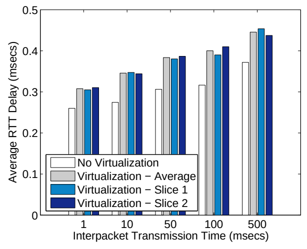


(b) FTP performance with 1GB file transfer.

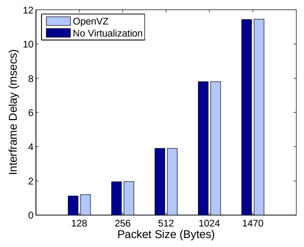
Fig. 4. Measurement of UDP throughput with varying packet sizes and file transfer time with FTP. For the UDP throughput measurement, channel rate is constant at 36Mbps and packet size is varied. For the FTP experiment, packet size is constant at 1024 and channel rate is varied.

varying offered loads and fixed frame size of 1024bytes in Figure 3(a) and its variance in Figure 3(b). Throughput obtained in the virtualized case are averaged over the two links. We observed that both below and above channel saturation there is no distinct difference in throughput with or without virtualization. This trend indicates that both virtualization platforms perform efficiently under saturation conditions. However, the variance in throughput with UML increases with offered load specially near and above saturation. Typically, this suggests that the OpenVZ platform benefits from tighter scheduling and lower overheads compared to UML.

To determine the effect of varying packet sizes, we fix the offered load to 40Mbps and transmission rate to 36Mbps, and vary packet sizes from 128bytes - 1470bytes. Figure 4(a) shows that for packet sizes less than and equal to 1024 bytes, UML has a significantly higher packet processing overhead which



(a) Average RTT delay.



(b) Interframe Space Vs Packet Sizes.

Fig. 5. Delay in different experiment scenarios. Minimum and average round trip time measurements are based on ping while interframe space measurements are based on difference in arrival times of packets at the receiver.

leads to a degraded performance. We attribute this degradation in performance with UML to the lack of support for virtualization in the host kernel.

Finally, we measure throughput performance of TCP by setting up a FTP transfer of a 1GB file with varying channel rates. Resulting file transfer times are as shown in Figure 4(b). For all channel rates, performance of UML is on par with OpenVZ and no virtualization due to the use of larger IP frames resulting in less performance overheads.

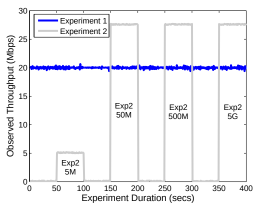
Thus for all three cases, we observe that OpenVZ has satisfactory performance, while UML's throughput performance suffers for small frame sizes.

4.2 Transmission Delay

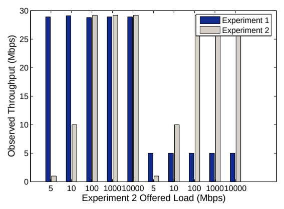
Delay and jitter are typically important for experiments that measure performance of real time systems or data. We measure delay and jitter performance in terms of distribution of delay across slices and distribution of delay overhead with varying packet sizes.

To measure the distribution of delay across slices we generate ICMP traffic (ping) across both slices and measure the round trip times (RTT). In Figure 5(a) we present the average RTT over an interval of 300 secs for varying packet arrival rates using OpenVZ. We plot delay measurements without virtualization, average delay across both slices, and delay across individual slices. The results show that in all cases, OpenVZ 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 smaller 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. Efficient buffer copying mechanisms enable OpenVZ to operate with little or no delay overheads, and it is safe for making temporal measurements across slices. A separate study [20] has shown performance degradation in UML under similar experiment settings.

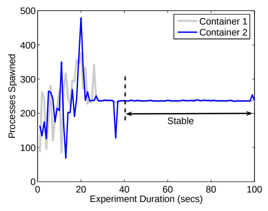
In order to evaluate the processing delay using OpenVZ, we measure the arrival time differences consecutive packets at the receiver with a constant sending rate. This difference in arrival times is also directly proportional to the delay [10]. We present this result as an average over $10,000$ consecutive frames of UDP traffic at 36Mbps in Figure 5(b). We repeat these experiments with various packet sizes. We observe that the delay increases with packet sizes due to increasing transmission times but there is little or no difference between the measurements with and without virtualization. Therefore we conclude that OpenVZ adds little overhead in packet processing and the overhead does not vary with packet size.



(a) UDP transient performance.



(b) Cross coupling between experiments.



(c) Process isolation test with *Fork bombs*.

Fig. 6. 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.

4.3 Slice Isolation

Isolation is an important requirement for a virtualized testbed since it directly determines the degree of repeatability achievable in a virtualized setting. Since OpenVZ has clearly outperformed UML in the previous experiments we will rule out UML for further experiments. To measure isolation we coin two performance measurement metrics: transient response and cross coupling between experiment.

We define transient response as the instantaneous change in throughput of an experiment running on one slice caused due to time varying change in offered load on another slice. To measure the transient response, we maintain the offered load for the experiment running on slice 1 at a constant value of 20Mbps and vary the offered load on slice 2 from 5 Mbps to 5 Gbps in steps. Results are presented in Figure 6(a). We see that there is little or no correlation in the throughput of the experiment running on slice 1 (over time) in response to the change in offered load in slice 2. Therefore we may conclude that OpenVZ provides reasonable isolation between slices.

We define 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 the experiment in slice 1 at constant values of 30Mbps and vary the offered load of the experiment on slice 2 from 5 Mbps to 10Gbps in steps. This experiment is then repeated with slice 1 fixed at 5Mbps. The throughput of each experiment averaged over 180seconds are as shown in Figure 6(b). We see that the results of the experiments in slice 1 are never affected by the change in offered load on slice 2 and therefore we concur that there is negligible cross coupling of experiments. It is important to note that these results are achieved without tweaking features of OpenVZ that allow the user to set custom cpu usage per slice.

Finally we present test results that measure process space isolation between the VPSs. As a part of these tests, each of the containers are triggered with fork bombs, and the number of processes spawned in each of the VPSs are as shown in Figure 6(c). We observe that the system quickly settles to an equilibrium where each of the containers share equal number of processes. Thus we observe that OpenVZ allows for successful containment of processes within each VM.

5 Related Work

There are several prior works that provide comparative analysis of virtualization platforms [18, 15, 11]. However, most of this work is in the context of server/machine virtualization. Authors in [18] study the scalability of four virtual platforms: Vserver [22], UML [6], Xen [16] and VMWare [8]. They perform a quantitative evaluation by measuring virtualization overhead, and isolation between VMs. They also measure startup time and memory occupancy of each virtualization platform. A similar study [11] has presented a comparative analysis of Xen, OpenVZ and VMWare Server using industry standard benchmarks for evaluating filesystem, network, multiprocessing and parallel processing performances. While these performance measures are important in our context as well, we concentrate more on the networking aspect of virtualization and platform suitability from a wireless testbed perspective.

The study in [20] discusses virtualization performance using UML by running two instances on a single Orbit node and isolating slices based on orthogonal channels. In our work, we extend this study by comparing the performance of OpenVZ based virtualization with the UML based scheme. Other previous wireless testbed [14] studies have more focus on the system architecture rather than features exported by the technology itself.

6 Conclusion and Future work

This study presents a comparison of qualitative features and performance which are useful from the perspective of a virtualized wireless testbed deployment. Our qualitative comparison shows that all forms of system virtualization could be used for virtualization of a wireless testbed. Measurements presented in the paper show that OpenVZ consistently outperforms UML in terms of system overheads, slice isolation and its performance is closest to that of the native non-virtualized system. This performance can be attributed to a tight virtualization mechanism and efficient approach to packet handling. Having selected Open VZ as the platform for Orbit virtualization, integration with the orbit framework and measurement library are the most important next steps. From a measurement standpoint comparison with Xen and Vservers on newer Intel chipset based machines are important future research items.

References

1. GENI design principles. *http://www.geni.net/*.
2. Kernel virtual machines. *http://www.linux-kvm.org/page/Main_Page*.
3. Madwifi driver. *http://www.madwifi.org/*.
4. OpenVZ instruction manual. *http://wiki.openvz.org/*.
5. Tcp/udp traffic generation tool. *http://dast.nlanr.net/Projects/Iperf/*.
6. A user-mode port of the kernel. *http://user-mode-linux.sourceforge.net/*.
7. VINI, a virtual network infrastructure. *http://www.vini-veritas.net/*.
8. VMWare player. *http://www.vmware.com/products/player/*.
9. A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In vini veritas: realistic and controlled network experimentation. In *Proceedings of SIGCOMM*, pages 3–14, New York, NY, USA, 2006. ACM.
10. G. Bhanage, R. Mahindra, I. Seskar, and D. Raychaudhuri. Implication of MAC frame aggregation on empirical wireless experimentation. In *IEEE Globecom 2009 Wireless Networking Symposium*, Honolulu, Hawaii, USA, Nov 2009.
11. V. Chaudhary, M. Cha, J. Walters, S. Guercio, and S. Gallo. A comparison of virtualization technologies for hpc. *Proceedings of AINA*, March 2008.
12. 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 *WCNC*, Mar 2005.
13. E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3), 2000.
14. M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stacky, K. Webby, J. Lepreau. Large-scale Virtualization in the Emulab Network Testbed. In *Proceedings of USENIX*, 2008.
15. S. Maier, D. Herrscher, and K. Rothermel. On node virtualization for scalable network emulation. In *Proceedings of SPECTS*, Philadelphia, PA, July 2005.
16. 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.
17. L. Peterson, S. Muir, T. Roscoe, and A. Klingaman. PlanetLab Architecture: An Overview. Technical Report PDN–06–031, PlanetLab Consortium, May 2006.
18. B. Quetier, V. Neri, and F. Cappello. Selecting a virtualization system for grid/p2p large scale emulation. In *Proceedings of EXPGRID workshop*, June 2006.
19. 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.
20. S. Singhal, G. Hadjichristofi, I. Seskar, and D. Raychaudhuri. Evaluation of UML based wireless network virtualization. In *Proceedings of NGI*, Poland, Mar 2008.
21. G. Smith, A. Chaturvedi, A. Mishra, and S. Banerjee. Wireless virtualization on commodity 802.11 hardware. In *Proceedings of Wintech*, pages 75–82, New York, NY, USA, 2007. ACM.
22. S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. *SIGOPS Oper. Syst. Rev.*, 41(3), 2007.
23. 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 *Proceedings of OSDI*, Boston, Dec. 2002.