Building an Operation Support System for A Fast Reconfigurable Network Experimentation Testbed

Alexander Poylisher, Yitzchak M. Gottlieb, Constantin Serban, Keith Whittaker, James Nguyen and Chris Scilla
John Lee, Florin Sultan, Ritu Chadha and Cho-Yu Jason Chiang U.S. Army CERDEC
Applied Communication Sciences Aberdeen Proving Ground, MD
Piscataway, NJ 08854

***Abstract*—We discuss in this paper the emerging need for an operation support system to support fast, reconfigurable, time-shared testbeds. We articulate the needs for building an operation support system for such testbeds in order to provide better utilization of testbed resources, enable testers to closely examine and analyze tests, streamline the process of test setup and execution, as well as enhance the efficacy of tests and the throughput of the testbed. In addition, the progress that we have made so far, our current research and development road map, along with foreseeable research challenges are also discussed.**

I. Introduction

Testing applications for deployment at scale is necessary to ensure that the applications behave as designed. A popular testing method, primarily in acceptance testing, is to deploy the software to a test environment, operate the system in some well-defined scenarios, and collect data on how (of if) the application performed. The key to a useful test is to have a test environment that mimics the deployment environment as closely as possible and to test as much application behavior as is practical in the time available. Two factors complicate such testing for applications expected to be widely deployed whose interactions with other instances of the same application and other applications is to be verified. First, large scale environments tend to be deployment environments, rather than test environments. Second, large scale environments are difficult to maintain, especially for testing purposes. These two factors lead to the development of reconfigurable virtualized testbeds, which provide large scale virtualized environments in which to deploy a test application, but do not require the resources an actual deployment would need.

Reconfigurable testbeds consist of some number of physical machines that support many configurable virtual machines (VMs) and are commected to a configurable interconnect. The virtual machines may be either fully virtualized or paravirtualized [1]. The interconnect between VMs, shown in Fig. 1, may be composed of hardware and software to emulate desired connectivity between the VMs [2]. To achieve higher fidelity with respect to preserving wireless network characteristics, the interconnet may be a network simulator with a high-fidelity simulation model. By taking advantage of the software-in-the-loop simulation capability [3], an operator could then test actual implementations, rather than emulations or models, in a virtualized, high-fidelity environment. In either case, traffic generated by the VMs is diverted through the interconnect.

Reconfigurable testbeds provide a location to which to deploy the test environment, but they still present challenges in how to deploy the environment and how to execute the test repeatably. Deploying a large scale test environment requires configuring a large number of hosts. Many of these hosts will be configured similarly. However, certain information about each individual hosts, for instances, its MAC addresses, will not be identical from host to host. Additionally, a heterogeneous host population is critical for interoperability tests. Deploying the test also requires configuring the interconnect. The interconnect configuration is not likely to be trivial when attempting to emulate network links with greatly different behavior such as radio links and high-speed wired links.

Once the testing environment is deployed the test must be run. Some tests do not require operator intervention, and the deployed environment must execute for some time. Others require an operator to interact with some part of the environment at particular times, such as to activate a program or enter a command. In either case the deployed test must record certain state, such as network traffic statistics, for later analysis. In order to allow greater confidence in test results, the test should be repeatable.

Given the difficulty in configuring and executing tests even in a virtualized environment, it is natural to avoid taking those steps. This implies that the testbed is dedicated to one application for long periods of time, leaving the testbed idle and preventing cost-efficient use of large numbers of computers. A reconfigurable testbed should be usable by as many simultaneous test scenarios as it has resources to support. A reconfigurable testbed should provide facilities for the easy restoration of a configured state so that it can be used for more than one test scenario without difficult and time-consuming configuration.

Given the difficulty in configuring and executing tests even

The research reported in this document/presentation was performed in connection with contract number W15P7T-08-C-P213 with the U. S. Army Communications Electronics Research and Development Engineering Center (CERDEC). The views and conclusions contained in this document are those of the authors and should not be interpreted as presenting the official policies or position, either expressed or implied, of the U. S. Army CERDEC, or the U. S. Government unless so designated by other authorized documents. Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

978-1-4673-3/12/$31.00 ©2013 IEEE

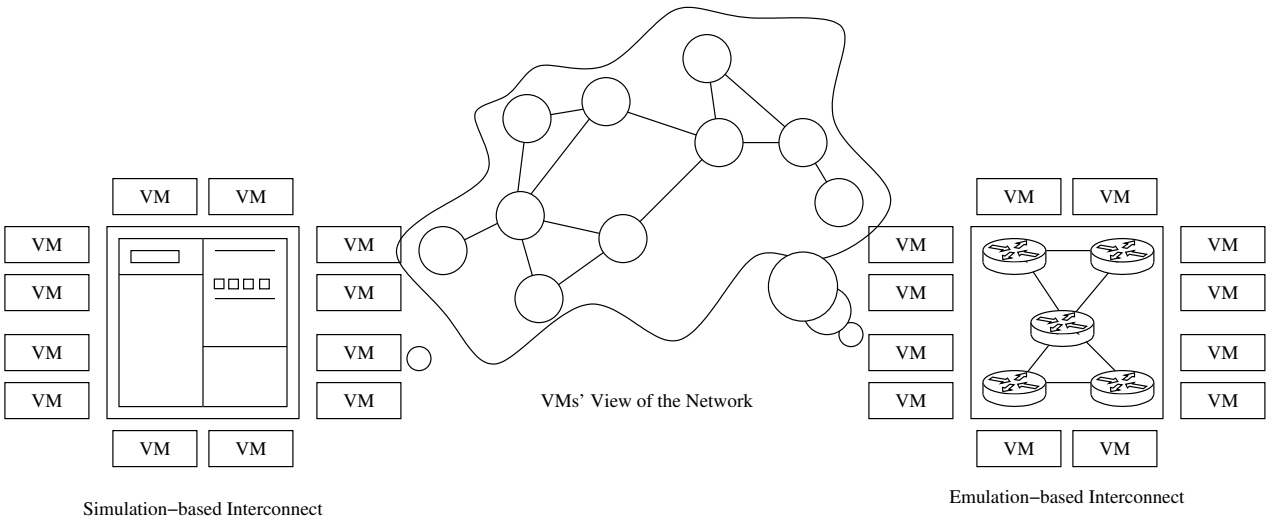

Fig. 1. Reconfigurable testbeds use simulated or emulsted network interconnects to produces the VMs' view of the network.

in a virtualized environment, it is natural to avoid taking those steps. This implies that the testbed is dedicated to one application for long periods of time, leaving the testbed idle and preventing cost-efficient use of large numbers of computers. A reconfigurable testbed should be usable by as many simultaneous test scenarios as it has resources to support. A reconfigurable testbed should provide facilities for the easy restoration of a configured state so that it can be used for more than one test scenario without difficult and time-consuming configuration.

These challenges mirror some of the challenges faced by users of early computers. Early computers, such as ENIAC, were programmed in hardware, so running a second program required reprogramming the entire computer. The development of computers that executed programs from input and stored program computers that executed programs from memory drove the development of operating systems that could execute any of a number of programs based on user input. Early computers were also expensive driving the desire to use as much of their time as possible. This led to developments in batch processing and various time-sharing operating systems where a given program would share resources for a time and then be removed from running. Given this precedent, we argue that reconfigurable testbeds should provide an operation support system to provide common services such as managing computational, storage, and network resources and managing testing scenarios.

The next several sections describe the capabilities required of a testbed operation support system (OSS) in resource management and test scenario management. Section IV will describe our current progress in creating TOSS, a testbed OSS for the VAN Testbed [2] and compare its capabilities to those offered by other user interfaces to testbeds. Finally, Section VI will describe our research direction and goals for enhancements to TOSS.

II. Test Scenario Management

Test scenario management includes the capabilities to configure the testbed according to a test scenario description and to execute the test scenario. Configuring the testbed requires managing the testbed's resources, see Section III, and configuring them to support the described test scenario. The test scenario description contains the information necessary to configure the testbed for the test scenario. It includes information describing each VM, such as the number and speed of processors that it requires, the operating system it executes, perhaps even the exact disk image with which the VM must be booted. Additionally it includes network information about the VM that directs the configuration of the interconnect, such as the IP addresses of each VM's interfaces. The description also includes actions that the VMs take at given times within the experiment, such as starting traffic or moving. Also, the scenario defintion must allow the operator to describe the properties of the network. The properties of the network may include time-dependent or event-dependent components, perhaps due to motion. In testbeds with simulation-based interconnects these events are configured in the simulation scenario, while in testbeds with emulation-based interconnects, these events may dynamically modify the interconnect to modify delays and to change connectivity.

Configuring the testbed to execute the test scenario is called *deploying* the scenario. Deployment requires mapping a test scenario's VMs to the testbed's resources. This typically involves selecting the physical machine on which a VM will execute, copying the VM's data to that machine, and configuring the VM according to the test scenario description. Deployment also configures the testbed's interconnect to support the apporpriate communication between VMs. Configuring the interconnect may include configuring hardware to limit communication to given paths, applying certain properties (such as high loss or high latency) to particular connections, and creating input files for network simulators. Withdrawing

a scenario is the operation in reverse; it is changing the configuration of the testbed's resources to the state they were prior to deployment. Once deployment has completed, the test can begin.

There are two models for starting a test scenario after deployment. The first is having the test scenario start implicitly immediately after deployment; the second is starting the test scenario explicitly. Starting a test explicitly allows the test operator to control events in the test more precisely since those events are defined from the start of the test scenario. Starting a test explicitly gives the operator the control to modify certain elements of the test after deployment. This enables test scenario reuse and, in conjunction with an explicit method to stop a scenario, provides an effective method to explore an appropriate test configuration without requiring an expensive deployment or withdrawl.

Executing the test scenario in a testbed that gives the operater strict control over the clocks of the interconnect and the VMs opens the possiblity of pausing a test scenario during its execution. While paused the VMs are not operational and the interconnect does not forward traffic, with the appropriate tools, the state of each of the VMs would be accessible to the operator, allowing for fine-grained monitoring of the VMs. This ability to pause and resume VM execution and to examine its state while stopped, similar to the ability of debuggers for programs, is valuable in security testing to examine the progress of an exploit and in examining the fine-grained behavior of systems deployed at a large scale.

The virtualized nature of test scenarios deployed in these testbeds enables managing the state of the testbed as whole as an aggregate of the state of the components. It is possible to save the state of a test scenario in the testbed and to restore the test scenario to a given state in the testbed. Saving the state of the testbed is useful for creating new test scenarios whose VMs are configured similarly, restoring a test scenario the state it was in at a previous point in time, and analyzing the data produced at a set of VMs during the execution of a test scenario. For each of these, slightly different data must be saved. For duplicating the configuration of the VMs, the state consists of each VM's current disk state at a given time. For restoring a test scenario to a given point in time, the testbed must save the complete memory and disk state of all the VMs. For analyzing data after a test scenario, the saved state is the disk state of the relevant VMs after it has synchronized its file system to the disk.

Saving the state of the entire testbed also requires saving the state of packets that cross the interconnect. In testbeds with simulation-based interconnects, saving the state of the network is equivalent to saving the state of the simulation or the entire memory and disk state of the simulation machine. In testbeds with emulation-based interconnects, the problem of saving network state is more difficult [4].

The ability to restore a testbed to a previous state and to pause the entire test scenario work synergisticly to provide the operator the ability to interact with a VM in a "virtual time." When the operator pauses a scenario, and all the VMs stop, s/he can save the complete state of a VM of interest and then resume or unfreeze that VM alone. The operator can the interact directly with the VM while the scenario is paused. When done, the operator can restore the state of the VM to the time at which the test scenario was paused. The operator can then resume the test scenario from the time at which it was paused.

III. Resource Management

A test scenario can only be deployed to a reconfigurable testbed if the testbed has sufficient resources available for the test. While the precise resources needed depend on the testbed and the test to be executed, it is possible to describe them in general. The resources of interest are compute servers, network capacity, and storage. Compute servers are the physical computers that will host the VMs. They are characterized by the number and type of available processors and the amount of available main memory. The number of available processors may be fractional if VMs are already associated with that computer, but do not require all the time available on a processor. The type of processor aggregates properties, such as processor architecture and processor speed, that make comparing the performance of different processors challenging.

Network capacity describes the traffic that the interconnect can support. This includes not just the capacity of the physical links of the interconnect, but also resources used to configure the interconnect for the deployed test scenario. For example, testbeds that use VLAN tags [5] to separate traffic from different VMs can only support a number of VMs equal to the number of VLAN tags supported by switches of the interconnect. In testbeds with simulation-based interconnects, the network capacity is a function of the number of VMs and the parameters of the simulation model. In addition, the processing speed of the simulation computer limits the size of a test scenrio that can execute in real time. This last factor can be mitigated by executing the VMs in simulated time. [6]

Storage refers to the amount of available non-volatile storage available to store the definition of the test scenario and the contents of the disk images of the VMs at deployment time and at subsequent times. Test scenario definitions were described above. The disk image of a VM is the full contents of the disk of the physical computer being virtualized, including the actual software that will be tested. For large numbers of substantially similar VMs it is possible to minimize the storage needed by using testbed-specific compression operations, such as copy-on-write snapshots.

A given testbed can support only a finite number of deployed scenarios at any given time. In order to maximize the use of the testbed it is necessary to carefully schedule the use of the testbed resources. In modern operating systems, preemptive process scheduling enforces a policy that allows a process to execute on the processor and virtual memory schedules that processes' access to main memory. Testbeds typically rely on the operator to schedule the use of the available resources.

IV. TOSS

This section describes TOSS our implementation of an OSS for the VAN Testbed [2]. An instance of the VAN Testbed is a testbed with a simulation-based interconnect that is composed of multiple compute servers each with multiple network interface cards. Each interface card connects the compute server to a different local area network for node management access and test traffic. In addition to compute servers, the VAN Testbed also has simulator hosts and a controlling host from which the operator controls the testbed. TOSS has three components, the Testbed Management Controller (TMC), Testbed Management Modules (TMMs), and Simulator Management Modules (SMMs). The TMC is a set of programs that execute at the controlling host and provide the user interface for the operator to manage resources and test scenarios. The TMMs execute at each compute server and configure the compute server for deployed scenarios. The SMMs execute on each simulation machine to control the simulator for a particular test scenario.

TOSS provides the operator tools to manage resources by registering available compute servers, available disk storage pools, and available simulator hosts. At deployment, the TMC maps each VM to a compute server, one or more VLAN tags, and management IP addresses. Mapping the set of VMs to available hardware based on the number and type of CPUs and the amount of RAM is a variation of the multidimensional knapsack problem, but for many deployments a simple greedy algorithm is sufficient. The TMC then configures the compute servers' access to the VM disk images, automatically configures the simluation network, including the network's switch, with VLAN tags to identify the VM from which a network frame was sent, and configures the simulator to forward traffic correctly between the VMs. Finally, the TMC commands the TMMs to start their respective VMs. For convenience, Testbed Operation Support System (TOSS) also configures operator access to the VMs via the management network, allowing the operator to interact with the VMs as if s/he had physical access to the host.

TOSS provides the user tools to register a test scenario description, to deploy and withdraw a test scenario, to start and stop a test scenario, and to save and restore a test scenario. The scenario definition language is a simple XML [7] format that defines for each test scenario what type of simulator is required, miscellaneous information about the simulation scenario, and information about each VM. The information about the VMs includes the name and IP address of each VM within the simulation as well as a base disk image to use for the VM. TOSS will create a copy of this base image for each VM and will configure this base image for each VM with additional information, such as the VM's IP addresses, during deployment. The base image copies for the VMs are made using copy-on-write snapshots to minimize the storage required.

TOSS supports saving copies of the disk and memory state of the VMs to storage on the TMC. The saved scenario can be redeployed, permitting the operator to quickly restore the testbed to a prior configuration. Since the VM disks are snapshots of a base image and since there tend to be many VMs that use the same base image, TOSS saves only the differences between the base image and the VM's disk. In addition, TOSS compresses the differences to further minimize the storage required. Once a scenario has been saved, it can also be reregistered if the TOSS databases become corrupt or inconsistent.

V. Other Testbed Types

Reconfigurable testbeds offer a unique deployment model suited for testing software in a virtualized environment. This model establishes requirements on an OSS that will not be provided by tools designed for other deployment models such as cloud computing and federated wide-area deployment testbeds such as PlanetLab [8] and the Global Environment for Network Innovations (GENI) project. Cloud computing enables client to contract for computing services from a provider allowing the client to access a scalable computing infrastructure quickly and inexpensively. Cloud computing models include:

- Infrastructure as a service (IaaS) granting clients access to physical or virtualized raw computing and network resources
- Platform as a service (PaaS) granting clients access to an existing deployment platform
- Software as a service (SaaS) granting clients access to fully configured servers executing a given set of software.

For testing new software, the IaaS model is the most applicable since existing deployment platforms and software does not yet exist. The advantage of cloud computing IaaS is its nearly infinite flexibility. This flexibility comes at the cost of requiring explict configuration of every piece of the infrastructure, such as hosts, switches, routers, and firewalls. While services may exist for some of this configuration, configuring the performance of the network connecting the infrastructure is typically not supported. This is understandable as the location of the physical nodes and their physical connections are specifically being abstracted from the user so that the provider can migrate and modify them as operationally required.

PlanetLab is a "geographically distributed overlay network designed to support the deployment and evaluation of planetary-scale network services." PlanetLab's major distinguishing characteristics are its geographic distrubtion and its focus on providing always available services. These two factors contribute to a management system that provides access to raw hosts and allows management services to allocate resources automatically and configure hosts. PlanetLab is effective at providing real deployments of novel, geographically-distributed services allowing those service to be actively tested and evolved with actual (as opposed to simulated) workloads. However, once again, this geographic distribution makes emulating or simulating a network with novel performance characteristics difficult if not impossible. Additionally, PlanetLab's distrubted nature makes management necessarily more complex than that used for centrally managed and centrally located testbeds.

The deployment model of reconfigurable testbeds, that is, that the very nature of the connecting network, not just its topology is a necessary component of experiments, makes them unique and requires that OSSs manage not just network capacity and network element configuration, but entire network configuration. This requirement has implications for resource allocation, for resource configuration, and for scenario definition. Resource allocation must take into account the resources that simulate or emulate, the network and the effect multiple test scenarios have on those resources. The OSS must support configuring the interconnect for various experiments and must support a defintion language that can describe the performance properties of the network.

VI. Future Directions

The current implemenation of TOSS is a practical and useful tool for configuring the VAN Testbed for a test scenario. It's current capabilities, described in Section IV, will be enhanced with additional capabilties for automating management of storage pools, for additional scenario scheduling capabilities, and for support of multiple, serial snapshots.

One of the current limitations of TOSS is the requirement that storage pools be created with a fixed size prior to use. This requirement stems from the underlying implementation of storage pools as LVM volume groups and the mapping between an entire storage pool and a compute server. However, it leads to fragmentation in the storage pools and the possibility of being unable to deploy a scenario due to the apparent, but not actual, lack of available storage for the VM disks. It seems that the obvious solution is to create storage pools per test scenario once the amount of storage needed for the scenario is known. However, practical considerations such as data compression available by exporting an entire volume group to a TMM, make this more challenging that it seems.

Another limitation of the current TOSS is its limited supported for saving scenarios. The current TOSS supports saving a single snapshot of the entire deployment in one of two modes: disk snapshot and disk and memory snapshot. However, supporting multiple snapshots enables repeatability and more complete test coverage. Consider an operator testing the applications that execute in a tactical military network for resilience to attacks. The operator would deploy a scenario to mimic as closely as possible the configuration of hosts in the tactical network. Once the experiment is configured and operational, the operator snapshots the system and continues operation. The experiment runs for some time with or without user input, and the operator once again snapshots the system. This pattern continues for some time. At some point, the operator stops the experiment and possibly snapshots the system again. At some later time, perhaps a few minutes, or a few months later, the operator discovers a need to run a small part of the test again with slightly different input. Instead of running the entire test again, which could take days, the operator simply uses a saved snapshot to restore the system to a given state within a previous experiment and continues the cycle of running and snapshoting the system. At some point the operator stops the experiment and, again, snapshots the system. This rerunning pattern is repeated multiple times. The operator may then compute any number of metrics over the saved snapshots. Finally, the operator archives the snapshots for future use.

The first challenge in supporting mulitple snapshots is the deployment of an efficient snapshot creation tool that can be used as a basis for creating entire system snapshots. The next challenge is to create a technology that will allow the snapshot creation system to create snapshots that are consistent across the entire experiment that can span thousands of virtualized (or para-virtualized) hosts. After snapshots exist, the next challenge is to create a system that can restore the state of an experiment to match a snapshot. Finally TOSS must address the challenges of managing (organizing and selecting for use) a large number of snapshots from a large number of hosts, as well as the efficient storage of large numbers of large binary files.

Another direction for future research an development is the enhancements to TOSS's scenario scheduling. Currently, TOSS scenarios are scheduled manually and explicitly. A scenario is deployed and withdrawn in response to a user command. Resources allocated to a scenario are held throughout deployment, and the scenario is started and stopped explicitly by the user. In operating systems terminology, test scenarios are not automatically preemptable. For test scenarios that can operate entirely autonomously, such as applications that can be completely scripted, and for tests that require operator input at specific times, it is possible to schedule not the execution of a scenario, but operator access to it. Operators would submit a test scenario and a deadline by which it should be complete. In addition, the operator would specify times at which the deployed scenario would be available for access. The times would be specified either as real world time or as time within the scenario. TOSS would then be responsible for executing the scenario up to the avaialability points, pausing the scenario, and storing it for access. With multiple scenarios available for deployment, the ability of the system to swap out one experiment for another without operator input is very similar to process and VM scheduling in an operating system. This allows more efficient use of testbed resources in that the testbed is constantly in use.

There are several challenges to supporting such a capability. The first is the definition of the exact need for a given deployment of a test scenario. For the same test, the operator may have different needs. For example, in some deployments the operator may want to debug the test, so the test cannot act autonomously. Deployments that require user interaction cannot be batch-scheduled indiscriminantly, but rather must be scheduled in concert with user availability. Modern operating systems can use user interaction as a trigger to schedule a process with which the user interacts, however, swapping in a program is much shorter than deploying a large scenario. Research in this area will focus on scheduling, effective methods of reducing deployment time, and methods for allowing TOSS to support a preempted deployment without fully withdrawing

the scenario.

VII. Concluding Remarks

In this paper, we present TOSS, a testbed operation support system that has been developed for the Virtual Ad hoc Network (VAN) testbed. TOSS has been shown a valuable tool and a concept that we believe will be embraced by the test and experimentation community. Even at the early research and development phase, TOSS has been shown to provide excellent value to developers, testers, and system evaluators alike. Many of the concepts and implementation of TOSS is generic, and hence with minimal modification and customization TOSS could be used for many different testbeds. We will continue the development of TOSS to facilitate the various testbed operations such that test throughput and test efficiency can be greatly enhanced.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and W. Andrew, "Xen and the art of virtualization," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, Bolton Landing, NY USA, 2003.

[2] A. Poylisher, C. Serban, T.-C. L. John Lee, R. Chadha, and J. C.-Y. Chiang, "A virtual ad hoc network testbed," *International Journal of Communication Networks and Distributed Systems*, vol. 5, no. 1/2, pp. 5–24, 2010.

[3] J. Youn, M. Jooyoung, S. Ma, and W. Lee, "Software-in-the-loop simulation environment realization using matlab/simulink," in *SAE 2006 World Congress & Exhibition*, Detroit, MI USA, 2006.

[4] A. Burtsev, P. Radhakrishnan, M. Hibler, and J. Lepreau, "Transparent checkpoints of closed distributed systems in emulab," in *Proceedings of the Fourth ACM European Conference on Computer Systems*, Nuremberg, Germany, 2009.

[5] *IEEE Standard for Local and metropolitan area networks—Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks*, IEEE Std.

[6] F. Sultan, A. Poylisher, J. Lee, C. Serban, C.-Y. J. Chiang, R. Chadha, K. Whittaker, C. Scilla, and S. Ali, "Timesync: Enabling scalable, high-fidelity hybrid network emulation," in *Proceedings of The 15th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM'12)*, Paphos, Cyprus, Oct. 2012.

[7] "Extensible markup language (xml) 1.0." [Online]. Available: http://www.w3.org/TR/xml/

[8] L. Peterson, M. E. Fiuczynski, and S. Muir, "Experiences building PlantLab," in *Proceedings of 7th USENIX Symposium on Operating System Design and Implementation (OSDI '06)*, Seattle, WA USA, Nov. 2006.