An Integrated Testbed for Virtual Ad hoc Networks

Pratik K. Biswas, Constantin Serban, Alex Poylisher, John Lee, Siun-Chuon Mau, Ritu Chadha, Cho-Yu J. Chiang
Applied Research, Telcordia Technologies, Inc.
{pbiswas, serban, sher, jolee, smau, chadha, chiang}@research.telcordia.com

Robert Orlando, Kimberly Jakubowski
U.S. Army CERDEC
{Robert.Orlando1, Kim.Jakubowski}@us.army.mil

Abstract— Testing of applications for ad hoc networks poses a special technical challenge due to the difficulty of conducting experiments in an ad hoc network environment at a scale larger than a few nodes. One approach is to conduct experiments in a testbed that can imitate an ad hoc network. This requires the development of technologies that enable multiple instances of unmodified application software on a set of hosts to communicate via a simulated network that behaves like a real ad hoc network. In this paper, we describe the testbed developed under the *Virtual Ad hoc Network (VAN)* project[1] for testing applications over ad hoc networks, with a special focus on network management applications. The testbed employs *Xen-based virtualization* to achieve resource scalability. The infrastructure for the testbed provides an integrated platform consisting of *virtual nodes running the actual software under test,* augmented with a *simulated network* environment. Our goal is to enable software testing over large-scale (500-1000 nodes) ad hoc networks using the *VAN Testbed.*

Index Terms— *Mobile Ad hoc Networks (MANETs), Virtual Ad hoc Network (VAN), emulation, software-in-the-loop (SITL) simulation, hybrid testbed, virtualization, virtual machines, simulated network, shadow nodes.*

I. INTRODUCTION

Mobile Ad hoc NETworks (MANETs) are characterized by their capability to enable networking without any infrastructure support. Specifically, there are no wires and no dedicated routers. As there are no wires, nodes use wireless radios with limited bandwidth and typically high loss rates to communicate; as there are no dedicated routers, every node participates in packet forwarding; since the location of a node is not fixed and nodes can enter and leave the network at any time, network topology is dynamic.

Testing and evaluation of the functional correctness and communications performance of distributed applications over dynamic MANETs present significant challenges with regards to feasibility and scalability. First, ad hoc network hardware, particularly for the custom-built, is typically experimental and available only in small quantities, so conducting evaluation of more than a few real nodes is either infeasible or impractical for cost reasons. Second, running large-scale evaluations for real ad hoc networks in a physical terrain is very costly in terms of logistics, therefore the number of tests that can be performed would be much less than adequate.

One approach is to conduct testing over a testbed in a laboratory environment, where the testbed behaves like a real ad hoc network to the applications. Our goal is to develop testbed technologies that enable testing and evaluation of any software application on any common operating system, given any ad hoc network scenario.

This paper presents the testbed developed under the *Virtual Ad hoc Network (VAN)* project to test applications over ad hoc networks, with a special focus on testing network management applications. The VAN technologies allow multiple application instances running on virtual nodes, possibly hosted on the same physical machines or on different ones, to send IP packets to each other via a simulated ad hoc network, as if each application instance is running on a real mobile node and the packets are transmitted over a real mobile ad hoc network.

Our main contribution for this endeavor has been to enable point-to-point and broadcast communications between Xen-based virtual hosts *via* an OPNET-based [18] simulated network. We have developed such capability by introducing a special kind of node emulation in which the IP protocol stack functions are split between Xen-based virtual hosts and simulated nodes, at the IP layer. Further, we have used Layer 2 packet forwarding with MAC address translation between the two parts of the split stack to provide increased performance of the testbed. The testbed achieves resource scalability through Xen-based virtualization. Finally, to facilitate the testing of different types of applications, we have also developed some capability for managing the resources of the virtual hosts.

The remainder of the paper is organized as follows: Section II discusses related work. Section III explains the problem background and relevant technologies. Section IV describes the network model under consideration. Section V presents the architecture and functionality of the proposed testbed, along

[1] 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.

with an introduction to the associated concepts and objectives. Section VI shows results with performance data for several application executions. Section VII concludes the paper with a summary and some future plans.

II. RELATED WORK

Related work can be grouped into two main categories: (1) virtualization, and (2) hybrid testbeds for wireless networks.

Virtualization has been applied to operating systems both commercially and in research for nearly thirty years. IBM VM/370 [1] made use of virtualization to allow binary support for legacy code. VMware [2] family of products, such as, VMware Workstation, VMware GSX Server, VMware ESX Server, etc., virtualizes commodity hardware, allowing multiple operating systems to run on a single host. All of these examples implement a *full virtualization* of (or at least a subset of) the underlying hardware, rather than paravirtualizing and presenting a modified interface to the guest operating system. LPARs [3] and Denali [4], on the other hand, use the *paravirtualization* approach to build an infrastructure for distributed systems. Xen [5] is a high performance resource-managed virtual machine monitor (VMM) (or Hypervisor) which provides an idealized virtual machine abstraction, thereby allowing multiple commodity operating systems to share conventional hardware in a safe and resource-managed fashion, but without sacrificing either performance or functionality. Xen can use either paravirtualization or hardware-assisted virtualization. Due to its flexibility and performance capabilities, we have used Xen in our testbed.

Hybrid infrastructures [6 - 14] have proved to be useful in validating networking techniques and conducting large scale experiments for wired/wireless environments. Emulab/Netbed (a descendant of Emulab) [6 - 8] is a network testbed that integrates emulators, simulators and live networks into a common framework under a common user interface. Experiments on Emulab/Netbed testbed [7] can combine real elements with simulated elements, each modeling different portions of a network topology in the same experimental run. The testbed can be used for comparisons of simulated, emulated, and wide-area scenarios. The EXperimental Testbed for Research Enabling Mobility Enhancements (EXTREME) [9] provides an experimentation facility that supports the testing of networking algorithms and technologies for wireless environments in a close-to-real scenario. Its design has been inspired by Emulab. EMWIN/EMPOWER [10, 11] is an IP-based scalable framework for mobile wireless emulation. With EMWIN, the mobility of the target mobile wireless network with a number of mobile nodes can be faithfully emulated in a wired network environment. EMPOWER is capable of assisting the study of both wired and wireless network protocols and applications. The open access research testbed for next-generation wireless networks (ORBIT) [12] is a radio grid testbed that has been developed for scalable and reproducible evaluation of next-generation wireless network protocols. The ORBIT testbed consists of an indoor radio grid emulator for controlled experimentation and an outdoor field trial network for end-user evaluations in real-world settings. WHYNET [13] is a large-scale hybrid testbed for heterogeneous wireless technologies (MANET, Wireless LAN, 3G Cellular, Sensors, UWB, etc.) that combines the realism of physical testing with scalability, flexibility and repeatability of simulations. A hybrid testbed for distributed sensor networks, consisting of real and simulated environments, has been used in [14] to conduct a variety of integrated tracking and surveillance-based experiments. Results from these experiments have shown that a large number of distributed and interacting sensor nodes, with different capabilities and operating in different environments, can be incorporated in high fidelity experiments to analyze the challenging aspects of the surveillance problem through realistic application scenarios. Two testbeds, one virtualized and one not, have been used in [15] to demonstrate the efficacy of a measurement-based QoS solution for opaque MANETs.

III. BACKGROUND

In this section, we explain the three key technologies that provide the backbone of this paper and have been used in our testbeds.

A. Network Simulation and Network Emulation

Two of the most commonly used experimental techniques for the design and validation of new and existing networking ideas and technologies are network simulation and network emulation. In *network simulation*, a program [18] models the behavior of a network either by reproducing the interactions between the different network entities (hosts/routers, data links, packets, etc) using mathematical formulas, or by replicating behaviour of real network components. Network simulation provides a repeatable and controlled environment for rapid prototyping and experimental evaluation. *Network emulation* [6, 7] is a hybrid approach that combines real elements of a deployed networked application, namely, the end hosts and protocol implementations, with synthetic, simulated, or abstracted elements, namely, the network links, intermediate nodes and background traffic. A fundamental difference between simulation and emulation is that while the former runs in virtual simulated time, the latter must run in real time. Another important difference is that it is impossible to have an absolutely repeatable order of events in any emulation, due to its real-time nature and often a physically-distributed computation infrastructure. *Hybrid testbeds* perform integrated network experimentation by combining real and/or emulated elements with simulated ones.

B. Software-in-the-loop Simulation

Software-in-the-loop (SITL) simulation is a methodology that utilizes a simulation-based approach to evaluate the effectiveness and scalability of real software as it is deployed in the field [16]. This methodology combines the use of unmodified compiled code from a real application with the simulation model. This results in high fidelity experiments as

there is no implementation-related inconsistency between the deployed and the in-the-loop software, which also happens to get tested more extensively in a close-to-real simulated environment. This methodology also eliminates the need for deriving and developing a model for the real software in the simulator, as the in-the-loop software integrates easily with the simulation model. To be effective, the software-in-the-loop solution needs to be platform-independent and neutral to the choice of the simulation engine. In our experiments, we model the MANET in a *simulator* and use Linux-based applications as the *software-in-the-loop*.

C. Virtualization

In computing, *virtualization* is a broad term that refers to the abstraction of computer resources. Virtualization makes it possible to run multiple operating systems (OSes) on the same computer at the same time. Virtualization can be grouped into two main types, i.e., platform virtualization and resource virtualization. *Platform virtualization* involves the emulation of whole computers, while *resource virtualization* involves the emulation of combined, fragmented, or simplified resources. Platform virtualization is performed on a given hardware platform by *host software* (a *control program*), which creates an emulated computer environment, a *virtual machine*, for its guest software. The *guest software*, which is often itself a complete OS, runs just as if it were installed on a stand-alone hardware platform. Typically, many such virtual machines are emulated on a single physical machine, where the number of the virtual machines is limited only by the host's hardware resources. Generally, there is no requirement for a guest OS to be the same as the host one. Three main approaches to platform virtualization are: *full virtualization*, *hardware-assisted virtualization* and *paravirtualization*. In the first, the virtual machine emulates enough hardware to allow an unmodified guest OS (one designed for the same CPU) to be run in isolation (e.g., Virtual PC, QEMU). In the second, the hardware provides architectural support that facilitates building a virtual machine allowing unmodified guest OSes to be run in isolation (e.g., Linux KVM, Xen, etc.). The second approach is an optimization of the first. In the third, the virtual machine does not necessarily emulate hardware, but instead (or in addition) the virtual machine monitor (VMM) offers a special API that can only be used by the modified guest OS (e.g., Denali, Xen). In this paper, we present the use of Xen paravitualization approach for hosting several *Linux-based* guest OSes on the same physical box, to run several instances of the network management applications.

IV. NETWORK MODEL

The network model used in the *VAN Testbed* implements the functionality of a network layer (OSI) and below. The network layer in our model consists of two sub-layers, i.e., the *mobile network layer* and the *mobile link layer*. In this model, packets originate at the IP layer. The IP packets are transferred to the *mobile network layer* and subsequently handed down to the *mobile link layer*. The mobile link layer is responsible for modeling TDMA transmission between two neighboring nodes within radio range. The mobile network layer is responsible for packet routing and end-to-end (multi-hop) forwarding. IP packets are encapsulated within the mobile network layer packets at the source node and are de-capsulated at the destination node. Packets arriving at intermediary nodes are forwarded at the mobile network layer level without reaching the IP layer. In this model, the IP layer is introduced to offer compatibility with the IP based networks, and it functions as a wrapper for the mobile network layer.

The network model supports two types of traffic: a) unicast traffic, destined for a single destination node; b) broadcast traffic, destined to all the nodes within radio range and which are assigned to the same mobile subnet as the source node. Nodes with multiple radio interfaces belong to multiple subnets; such nodes play the role of gateways.

Using the above model, we have developed several network scenarios designed to replicate the typical numbers of interacting nodes, subnet and channel allocations, as well as real life mobility trajectories for these nodes. The number of mobile nodes that are part of a network scenario, as well as their maximum traffic rates have been limited by the *VAN Testbed* constraint to have the simulation run no slower than real time. A more detailed description of some network scenarios is provided in Section VI along with the applications tested over these scenarios.

Figure 1 shows the model of a network node, and the flow of packets between its layers from a source to a destination node.

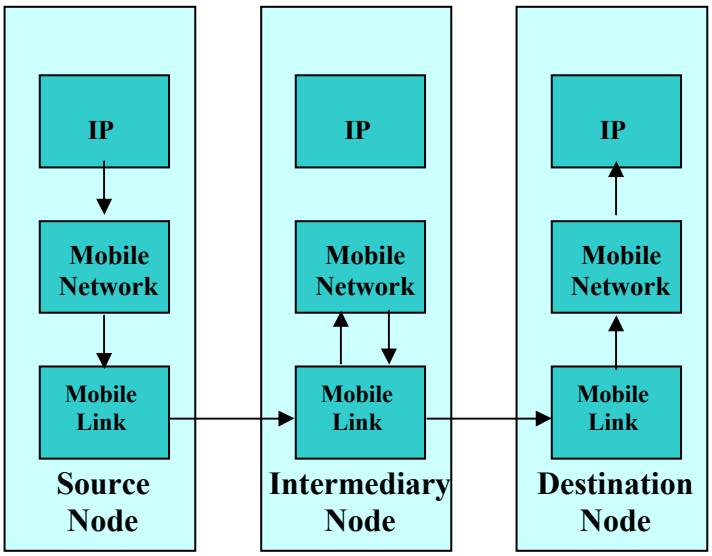

Figure 1: Mobile Node Model and Cross-Layer Packet Flow

V. A VIRTUAL AD HOC NETWORK (VAN) TESTBED

Testing of real applications over dynamic MANETs poses a significant scalability related challenge. One approach to address this challenge is to conduct testing on a testbed that behaves like a real MANET. We propose a *Xen-based*

virtualization [13] of the MANET. Accordingly, we have developed a *Virtual Ad hoc Network (VAN) Testbed* that applies the SITL simulation methodology to integrate a simulated MANET with the *in-the-loop* real network management application instances, running on Linux-based virtual machines. The testbed virtualizes network resources on physical host machines. With Xen virtualization, a thin software layer known as the *Xen Hypervisor* is inserted between the server's hardware and the operating system. It provides an abstraction layer that allows each physical server to run one or more *virtual servers*, effectively decoupling the operating system and its applications from the underlying physical server. The Xen Hypervisor is a powerful open source industry tool for virtualization. It can use *paravirtualization* as well as *hardware-assisted full virtualization*. It offers a powerful, efficient, and secure feature set for virtualization of several mainstream CPU architectures, and supports a wide range of guest operating systems including Linux, Solaris, various BSD variants and Windows (unmodified). A Xen system is structured with the Xen Hypervisor as the lowest and most privileged layer. Above this layer are one or more guest OSes, which the Hypervisor schedules across the physical CPUs. The first guest operating system, called *domain 0 (Dom0)*, is automatically booted after the Hypervisor. Additional guest OSes, called *domain Us (DomUs)*, are started from *Dom0.* DomUs are managed from Dom0.

A. Acronyms, Concepts and Objectives

This section provides a collection of terminologies, concepts and objectives that are essential for describing and implementing the architecture and functionality of the proposed *Virtual Ad hoc Network (VAN) Testbed.*

Definition 5.1: Network virtualization represents the process for combining hardware and/or software to create a *virtual network* that behaves like a real network. A *virtual ad hoc network (VAN)* virtualizes a MANET.

Definition 5.2: An *emulated node* represents a node that imitates a real node, emulates its protocol implementation, and runs its operational applications. As part of this *emulation*, the protocol stack of an emulated node can be split into two parts at some layer of its protocol stack, where the upper part of the protocol stack is implemented by actual OSes, e.g., Linux and Windows, and the bottom part is implemented by simulation models of lower layers.

Definition 5.3: A virtual machine (VM) represents a node in the *VAN Testbed*, hosted on a DomU, which can run an instance of the *emulated node* including only the upper part of the protocol stack.

Definition 5.4: A simulated network represents a collection of software components that model the functionality of a real network and its devices. For example, a *simulated network* can be implemented using a network simulation tool like OPNET.

Definition 5.5: A split protocol stack represents a protocol stack where the higher layers of the stack are implemented in the *virtual machines* and the lower layers of the stack are implemented by the *Simulated Network*. The point at which the stack is split is called the *splitting layer*.

Definition 5.6: A shadow node represents a node in a *simulated network* that simulates the functionality of certain ad hoc network communication protocols. In this paper, the shadow node is assumed to implement the part of split protocol stack at and below the *splitting layer*.

Our *goal* has been to develop a testbed that meets the following *objectives.*

Objective 5.1: The testbed shall support *emulation* of a MANET by using a *SITL simulated network*.

Objective 5.2: The testbed shall support *emulated nodes* where the part of the protocol stack above and including the *splitting layer* runs on a *virtual machine* (DomU), while the part of the protocol stack below and including the *splitting layer* runs on a *shadow node*.

Objective 5.3: The testbed shall support IP as the *splitting layer*.

Objective 5.4: The testbed shall support both unicast and broadcast transmissions over the *simulated network*.

Objective 5.5: The testbed shall support testing of any unmodified, distributed third party application running over *simulated networks*.

Objective 5.6: The testbed shall allow *remote management* of its resources.

B. Core Functionality

Our goal is to create a MANET testbed that enables testing and evaluation of any software application on supported operating systems for any MANET scenario, without necessitating changes to the application under test. The core functionality of the testbed involves the following major operations.

I. Transparent Forwarding – The testbed facilitates transparent communication among *virtual machines* (DomUs), using IPv4 addresses. Messages exchanged between applications running on different *virtual machines* (DomUs) will be forwarded through the Simulator Machine, without the applications being aware of them.

II. Local Stack Management – The testbed provides a mechanism for the applications to retrieve information available in the *simulated network* at their corresponding shadow node. Typical information provided includes the current mobile node coordinates and radio characteristics, such as the SNR. This communication is meant to assist applications that are designed to interact with the layers of the *split protocol stack* below the *splitting layer*. In the current implementation, this interaction is SNMP-based. The descriptions of the Local Stack Management and its usage are beyond the scope of this paper.

III. Remote Testbed Management – The testbed allows remote management of applications through interfaces that are different from those used for *transparent forwarding*.

C. Architecture

The *VAN Testbed* supports the *emulation* of a MANET by using a *SITL simulated network*. Accordingly, our testbed can currently integrate *emulated nodes* and a *simulated network*

into a common framework under a common user interface. An *emulated node* in our testbed employs a *split protocol stack*, with IP as the *splitting layer*. Experiments on our testbed can therefore combine *virtual machines* (DomUs) with *shadow nodes*, in the same experimental run. This section presents the system architecture of the *VAN Testbed*.

Figure 2 provides an overview of the physical architecture of the entire testbed. The following is a brief description of its two main functional components.

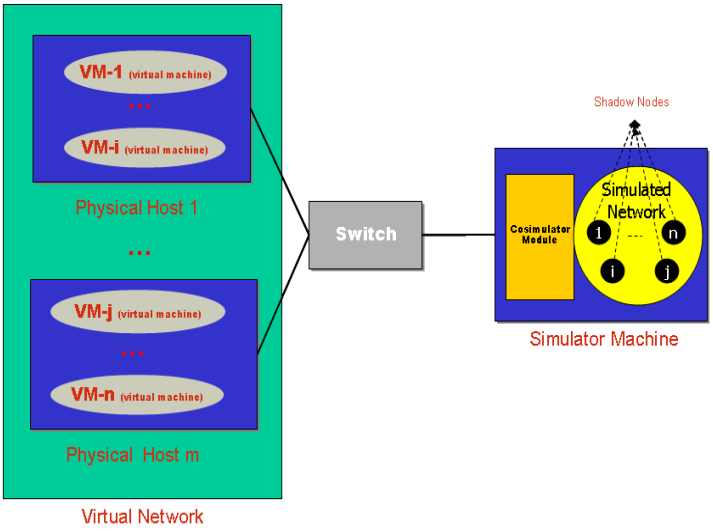

Figure 2: Physical Architecture of the *VAN Testbed*

Simulator Machine: This is a physical machine that is dedicated to running the *simulated network*. The *simulated network* hosts several *shadow nodes* (1..n) and comes with a Cosimulator Module (COSIM) that integrates it with the rest of the testbed (Virtual Network). This machine also runs the WinPCap daemon and the Windows firewall. Currently, the testbed uses OPNET Modeler 11.5 [18] for the simulated network. This is executed on a dual-core, 2-processor machine with 2G of RAM per 2 GHz processor, running Windows.

Virtual Network: The testbed uses a Xen-based virtualized network to host the applications to be tested. The entire Virtual Network is hosted on several Intel quad-core, 8 CPU, 8G physical host machines (1..m). On each such physical host machine, Xen Hypervisor is installed at the lowest layer; NetBSD-current is installed as Dom0 and Linux (Fedora Core 8) as DomUs. There are several *virtual machines* or *VM*s (DomUs), named VM-1 through VM-n, that are running on these physical machines (1..m), each with 256M of RAM. Each *VM* (DomU) is on its own subnet (in the 10.0 address space), and can indirectly talk to other *VM*s (DomUs). Each *VM* (DomU) is provided with a separate management interface, using an address from the 10.128 address space, through which graphical applications can also be run. Each Dom0 has two interfaces; one (Dom0-bnx0) connects it with the rest of the testbed through an internal LAN and the other (Dom0-bnx1) provides access to the outside world. For each *VM* ($DomU_i$, 1 <= i <= n), Xen creates two new pairs of *connected virtual ethernet interfaces*, where one end of each of the two pairs, i.e., ($DomU_i$-eth0) and ($DomU_i$-eth1) are within the $DomU_i$ and the other end, i.e., (Dom0-xvifi.0) and (Dom0-xvifi.1) exist within Dom0. There are 2n bridges (Br_i, 1 <= i <= 2n) in Dom0, one for each pair of connected interfaces.

Figure 3 and **Figure 4** provide overviews of the high-level and detailed architecture (bridges, interfaces, sub-networks, etc.) of the Virtual Network, as hosted on a single physical machine, i.e., for the base case of the Virtual Network with a single physical host machine.

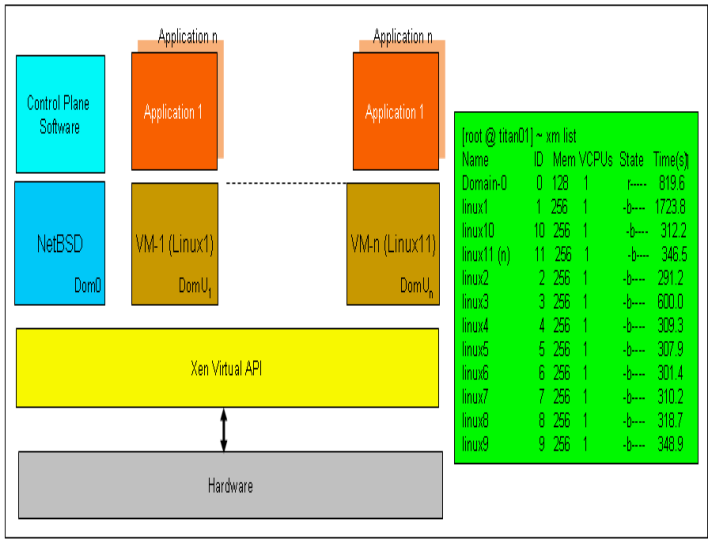

Figure 3: High-level Architecture of a Single Physical Host in the Virtual Network

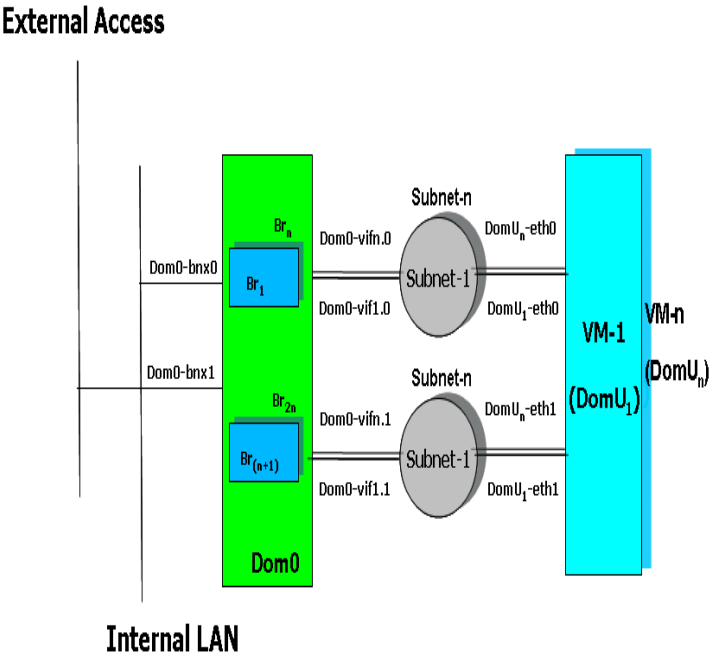

Figure 4: Detailed Architecture (Interfaces & Bridges) of the Virtual Network (Single Host)

1) The Simulator Machine and Relevant Processes

The main task of the Simulator Machine is to execute the simulation scenario corresponding to the desired network behavior, and to route the IP communication between the *virtual machines* (DomUs) through this simulation. In order to enable this functionality, the Simulator Machine executes a number of processes and services, as depicted in **Figure 5**, including the simulation process, the SNMP agent process, and the firewall process.

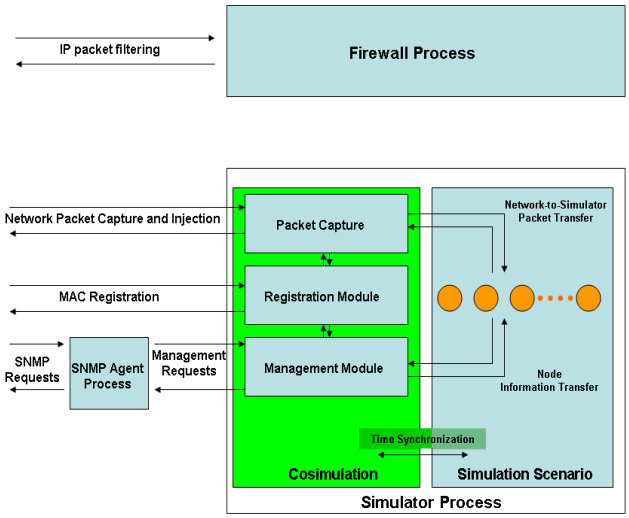

Figure 5: Processes and Servers on the Simulator Machine

The *simulation process* is composed of the following subcomponents:

- The *Simulation Scenario* models a number of nodes and their Layer 3, 2, and 1 communication stacks. The scenario executes for a given period of time, simulating given trajectories for each individual node.
- The *Cosimulator Module (COSIM)* provides a framework for launching and controlling the execution of the simulation. The main purpose of the Cosimulator is to transfer real communication packets into the simulation, which is achieved through the following components:
    - The *Packet Capture Component* is a server based on WinPCap that listens to, and sends out Layer 2 packets originating in, and destined for the *VM*s (DomUs). The Packet Capture Component is also responsible for transferring packets to and from the simulator.
    - The *Registration Module* is a server that maintains a mapping between the IP addresses of the *VM*s (DomUs) and the physical MAC address of the machine hosting them. Physical hosts hosting *VM*s (DomUs) register to this module prior to starting the simulation scenario. The MAC address corresponding to the *VM* (DomU) is used by the Packet Capture Component to correctly route the packets to their destination.
    - The *Management Module* is a server that exports network information that is modeled in the simulation to interested parties. Typically, *VM*s (DomUs) are querying information related to their corresponding shadow simulation nodes.

  The *Cosimulator Module* is also responsible for advancing the simulation scenario in real time, or at a pace suitable for the *virtual machines* (DomUs).
- The SNMP Agent Process is a SNMP agent that serves as a front end to the Management Module.
- The Firewall Process is responsible for dropping the Layer 3 (IP) packets that are intercepted by the Packet Capture Component at Layer 3, to prevent possible duplication and improve performance. The firewall drops all incoming IP packets arriving at the simulation host but destined for an IP address other than that of the Simulator Machine.

2) Transparent Forwarding

A virtual hub-and-spokes network is set up between *virtual machines* (DomUs) with the simulator stand-in as the hub. Messages can be exchanged between applications running on any two *VM*s (DomUs), hosted on the same or different physical machine(s), via the Simulator Machine. Indirect (virtual) point-to-point or broadcast communication between any two *VM*s (DomUs) is accomplished through the use of PF (Packet Filter) rules on Dom0, a packet capture daemon (WinPCap), and Windows Firewall on the Simulator Machine along with the MAC address registration process involving *VM*s (DomUs) and physical hosts. All *VM*s (DomUs) register the mapping between their IP addresses and the MAC address of the physical machine hosting the *VM*s (DomUs) with the WinPCap daemon. MAC address registration and translation are required as the WinPCap daemon processes the incoming and the outgoing frames at Layer 2 of the protocol stack, thereby routing at the MAC layer and not at the IP layer. PF (Packet Filter) rules are used on Dom0, to filter out the incoming interfaces, such that all outgoing traffic from any *VM* (DomU) destined to another *VM* (DomU) is first forwarded towards the Simulator Machine. The WinPCap daemon of the Cosimulator Module on the Simulator Machine subsequently routes copies of the frames via the *shadow nodes* in the OPNET Simulator, while the Firewall drops the actual traffic. Finally, the WinPCap daemon, after receiving back the frames from the OPNET simulator, inserts the appropriate destination MAC address corresponding to the IP address of the destination *VM* (DomU) in each frame (based on the dynamic mapping table), and then sends them to the destination *physical host*.

Figure 6 provides the details of this information flow. This figure describes the travel path of a frame (Layer 2 packet flow) from VM-1 (DomU) of Physical Host 1 to VM-n (DomU) of Physical Host m. The path consists of 4 segments, labeled (1) through (4). Segment (1) shows packet flow between physical machine hosting VM-1 (DomU) and the Simulator Machine hosting the Cosimulator Module (COSIM) & the simulator. Segment (2) shows packet flow between the WinPCap daemon (inside COSIM) and the OPNET simulator

inside the Simulator Machine. Segment (3) shows packet drop resulting from packet flow between the WinPCap daemon (inside COSIM) and the firewall. Segment (4) shows packet flow between the Simulator Machine and the physical machine hosting the destination *virtual machine*, i.e., VM-n (DomU). Black dotted lines point to packet contents at different segments of the path. In this path, the content of the frame changes only in segment (4) due to MAC address substitution inside the WinPCap daemon of the COSIM.

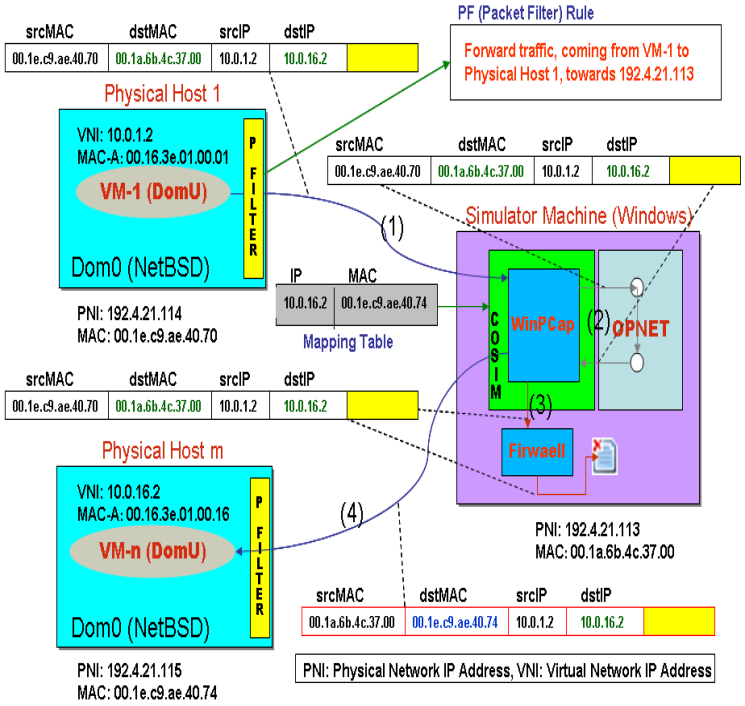

Figure 6: Information Flow between any two *Virtual Machines* (VM-1 and VM-n) for Transparent Forwarding

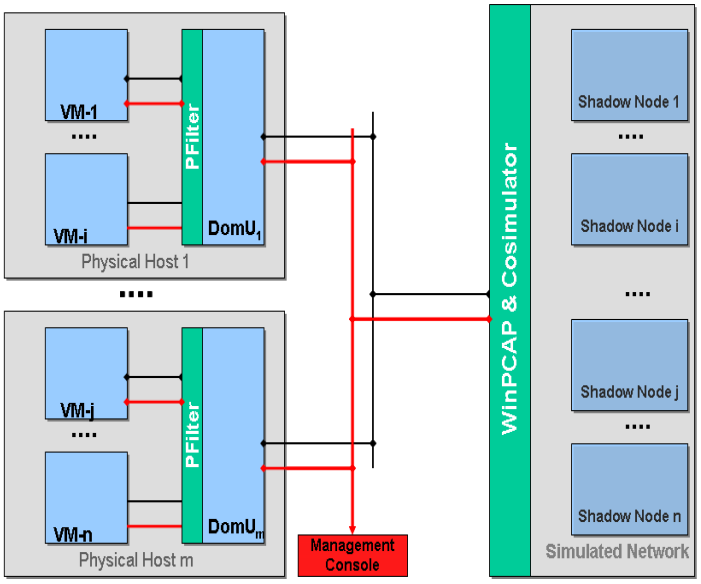

Figure 7: Overview of *VAN Testbed* Management

3) Remote Testbed Management

The *VAN Testbed* allows users to control and manage its resources. Users can access *VM*s (DomUs) and the Simulator Machine and manage the process of testing. Each *VM* (DomU) provides two network interfaces, one for *transparent forwarding*, and the other for management purposes. The *management interface* is different from the network interface used for *transparent forwarding*. It is used to access the *virtual machine* externally for command line interface, file transfer, application installation, graphical user interface (GUI) support, etc. **Figure 7** displays the two interfaces.

VI. VAN TESTBED APPLICATIONS AND RESULTS

In order to assess the effectiveness and performance of our testbed implementations, we have run a number of off-the-shelf communication-oriented applications. These applications fall into the following categories.

A. Network Management Applications

DRAMA (Dynamic Re-Addressing and Management for the Army) [16, 17] represents a policy-based MANET management system. DRAMA is composed of autonomous agents executing on, and managing each mobile node according to a set of policies. The most important characteristics of DRAMA are its scalability, low-bandwidth consumption, survivability, and autonomy and self-organization. DRAMA agents organize themselves into a management hierarchy that helps disseminate policies throughout the network, and propagate management information between various agents.

DRAMA represents a good candidate for execution over our *VAN Testbed* due to several of its salient features. First, DRAMA's management hierarchy is self-organizing, and it reflects the current network connectivity. Second, DRAMA has small bandwidth requirements, making it suitable for the type of mobile ad-hoc networks that we simulate. Third, DRAMA uses both broadcast communication (for the discovery of one-hop neighbors), and unicast messages for establishing the management hierarchy and other activities, thus providing a comprehensive and realistic application designed to run in MANET environments.

In order to evaluate DRAMA over the *VAN Testbed*, we have used a simulated network composed of 21 nodes. The testbed had been configured to execute on 3 physical machines, each hosting 7 *virtual machines* (DomUs), each hosting a Linux FC8 OS. Each *VM* (DomU) has been configured to execute a DRAMA agent. The network simulation scenario had been configured with 21 nodes, each corresponding to a *VM* (DomU). These nodes were organized into five subnets reflecting the relative node movements, where an entire subnet moves in a single compact formation. **Figure 9** shows the animation display providing the real time position of each node in the simulation. Subnets are depicted as compact groups of red nodes. These subnets move along the green arrowed lines. Subnets 2 and 3 are each composed of 7

nodes advancing eastwards; subnets 4 and 5 are composed of 3 nodes each, initially advancing westwards. Node 1 represents a stationary node assigned to subnet 1. One node in each of the subnets 2-5 is also assigned to subnet 1, to act as a gateway for its corresponding subnet. Red circles depict the radio range of node 1, as well as the ranges of subnets 4, and 5 respectively.

Figure 10 displays the DRAMA management hierarchy formed as a result of messages exchanged over the *VAN Testbed* during the network scenario. The Drama instance executing on node 1 is depicted in red, representing the root of the hierarchy. The blue nodes directly connected to node 1 represent the gateways for subnet 2 and 3, while the green nodes under them represent the rest of the nodes in subnets 2 and 3. The hierarchy connectivity reflects the direct communication range between various nodes, as observed in the simulation animation display. The two separate, unconnected hierarchies at the bottom of this display reflect the partitioning of the network, due to subnets 4 and 5 being out of range. In the course of the scenario, when subnets 4 and 5 move within radio range of subnets 2 and 3 respectively, DRAMA's periodic broadcasts get forwarded, and the DRAMA hierarchy becomes fully connected.

B. Network Tools

In order to assess the modeled *network connectivity*, *communication latency*, and *drop rates*, we have used the standard *ping* command between different pairs of *VM*s (DomUs). Ping request and reply packets, traveling through the simulated network, become subject to the modeled latency and drop rates. When node A pings node B, and the two nodes are disconnected, the ping output reflects no *ping echo reply* packets arriving back at node A, and the reported packet drop rate becomes 100%. When a multi-hop path between nodes A and B can be found, *ping echo reply* packets start arriving at node A, displaying a latency proportional with the number of hops between the two nodes. When the two nodes are in radio range of each other, the latency is minimal and the packet drop rate becomes zero.

Figure 11 shows the result of executing ping when two nodes transition from a multi-hop route into direct radio communication. The latency of communication decreases from about 500 ms to 200 ms RTT, thus accurately reflecting a roughly 100 ms modeled latency for a small packet transmission between two neighboring nodes.

C. Multimedia Applications

In order to evaluate the behavior of multimedia applications over our model *simulated network*, we have run an off-the-shelf *VIC* program over two *virtual machines* (DomUs) in the our testbed. *VIC* video tool is a real-time, multimedia application for **vi**deo **c**onferencing over the Internet, developed at UC Berkley/LBNL. VIC represents a good multimedia test application as it offers fine control over the amount of information sent over the network, and the type of transfer protocol employed, thus becoming suitable for use in mobile ad-hoc networks. Additionally, it offers a free, open source implementation for most major operating systems.

D. Benchmarking Applications

In order to evaluate the correctness of our *VAN Testbed* solution, we have attempted different types of communication (unicast, network broadcast, local broadcast traffic) as well as different protocols, such as TCP, UDP, ICMP, and raw IP datagrams successfully.

In order to evaluate the overhead introduced by our solution, we have tried to separate the latency and bandwidth factors modeled by the network scenario from the latency and bandwidth caused by the testbed. We have achieved this separation by creating an ideal network model that immediately forwards every single packet to its destination, without introducing any delay and drop rate. **Figure 8** presents the average latency for packets transmitted between two *virtual machines* (DomUs) in the presence of moderate traffic (up to 50 Mbps). The two *VM*s (DomUs) were located on different Dom0s, and the latency was averaged over 10,000 packets.

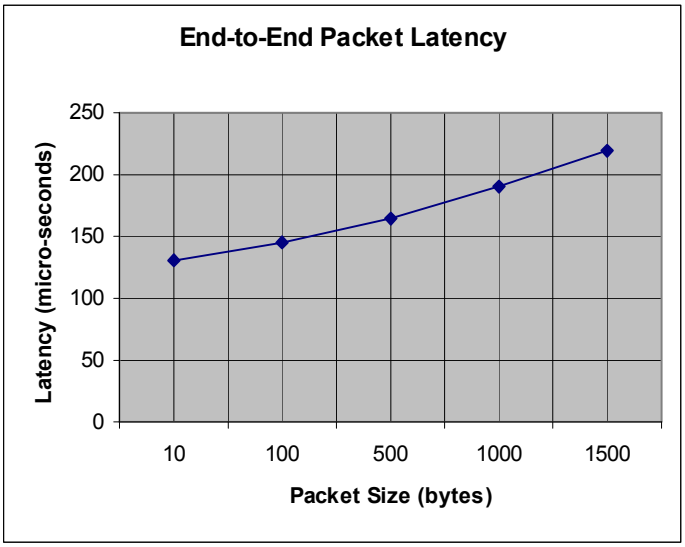

Figure 8: *VAN Testbed* Latency Overhead

Under such conditions, the observed overhead is between 130 and 220 microseconds. This latency is due to the following components: i) t_v, *VM* (DomU) to Dom0 latency (at both the sender and receiver sides), and ii) t_r, Dom0 to Simulation-process latency (at both sender to simulation and simulation to receiver sides, including two real Ethernet transmissions). The measured t_v component of the latency amounts to a value of about 25 microseconds, thus t_r dominates the overall latency.

In a different series of experiments, we have measured the overhead latency for different traffic loads. In addition to the moderate traffic above, the latency measured for 1,000-byte packets at the rate of 80 Mbps was 330 microseconds, and at the rate of 120 Mbps, the latency was 510 microseconds.

Even though the above absolute values might indicate a non-negligible latency, we believe that these values are relatively insignificant compared to the latencies experienced or anticipated in the MANETs under evaluation.

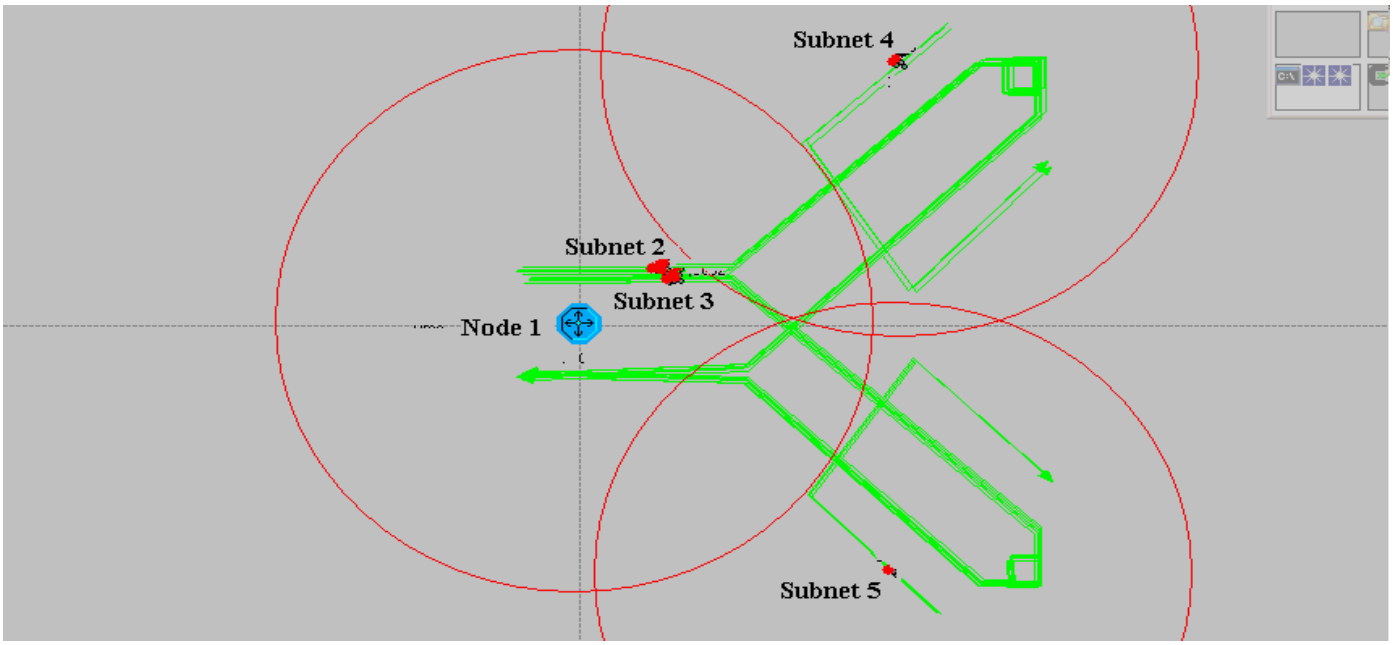

Figure 9: 21-node Network Scenario

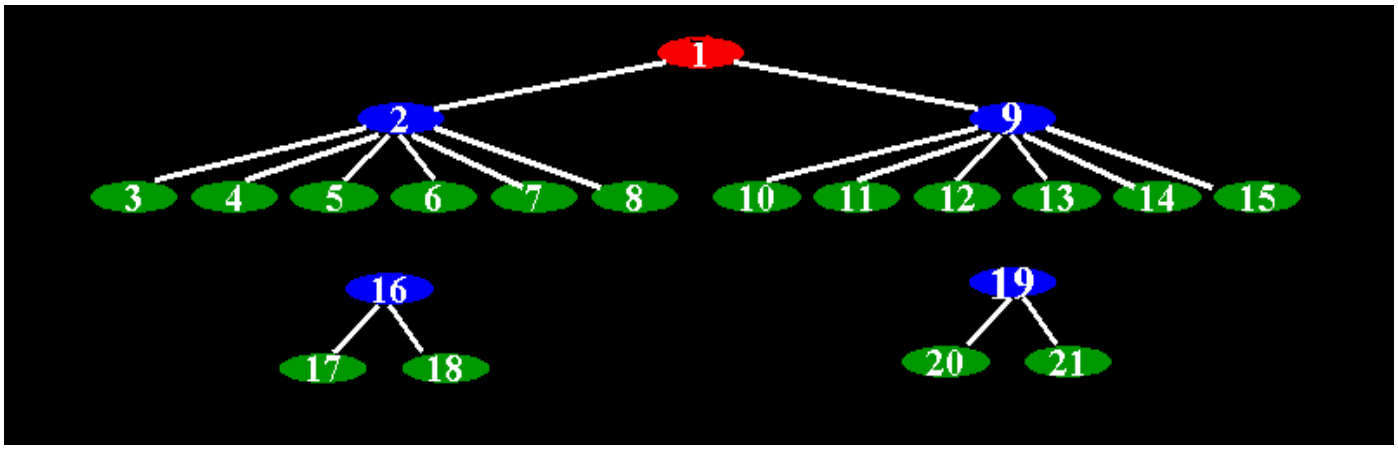

Figure 10: DRAMA Management Hierarchy

```
64 bytes from linux65 (10.0.65.2): icmp_seq=19 ttl=62 time=506 ms
64 bytes from linux65 (10.0.65.2): icmp_seq=20 ttl=62 time=414 ms
64 bytes from linux65 (10.0.65.2): icmp_seq=21 ttl=62 time=427 ms
64 bytes from linux65 (10.0.65.2): icmp_seq=22 ttl=62 time=472 ms
64 bytes from linux65 (10.0.65.2): icmp_seq=23 ttl=62 time=206 ms
```

Figure 11: Ping Command Output

VII. CONCLUSIONS

We have described a hybrid testbed that employs the software-in-the-loop simulation methodology for testing applications that can run on real MANETs, addressing the feasibility and scalability related challenges of testing and evaluation over MANETs. We have developed a Xen-based virtualized architecture for this hybrid testbed, where the applications run on multiple *virtual machines* (DomUs) and can communicate via the *simulated network*. The testbed facilitates message exchanges between the virtual and simulated worlds. We have implemented indirect point-to-point and broadcast communications through the use of packet filter (PF) rules, a packet capture daemon and a MAC-address based registration & translation mechanism. The testbed provides facilities for remote management of its resources. We have conducted experiments in the testbed; results demonstrate the applicability of the testbed for testing such applications. Preliminary results indicate that in the context of MANETs, the virtualized architecture can be advantageous as it not only reduces hardware considerably but also meets or exceeds our performance expectations.

Our future plan is to integrate the current *VAN Testbed* with emulated networks and real network devices in a lab environment. Our eventual goal is to establish benchmarks for running large-scale experiments on performance and QoS measurements in virtualized environments.

ACKNOWLEDGEMENTS

We would like to thank the Office of the Secretary of Defense (OSD) and U.S. Army CERDEC for supporting the work and providing valuable feedback.

REFERENCES

[1] Seawright, L., MacKinnon, R.: VM/370 - a study of multiplicity and usefulness. IBM Systems Journal, pp. 4-17 (1979)

[2] Devine, S., Bugnion, E., Rosenblum, M.: Virtualization system including a virtual machine monitor for a computer with a segmented architecture. US Patent, 6397242, (1998)

[3] Borden, T.L. et al.; Multiple Operating Systems on One Processor Complex. IBM Systems Journal, vol.28, no.1, pp. 104-123 (1989)

[4] Whitaker, A., Shaw, M., and Gribble, S. D.: Denali: Lightweight Virtual Machines for Distributed and Networked Applications. TR 02-02-01, University of Washington (2002)

[5] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. Proceedings of SOSP'03, New York (2003)

[6] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An Integrated Experimental Environment for Distributed Systems and Networks, Proceedings of the 5th Symposium on Operating Systems Design & Implementation (OSDI), pp. 255-270, Boston, MA (2002)

[7] Guruprasad, S., Ricci, R., Lepreau, J.: Integrated Network Experimentation using Simulation and Emulation. Proceedings of the First International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM '05), pp. 204-212, Trento, Italy (2005)

[8] Hibler, M., Ricci, R., Stoller, L., Duerig, J., Guruprasad, S., Stack, T., Webb, K., Lepreau, J.: Large-scale Virtualization in the Emulab Network Testbed, Proceedings of the 2008 USENIX Annual Technical Conference, Boston, MA, (2008)

[9] Portoles-Comeras, M., Requena-Esteso, M., Mangues-Bafalluy, J., Cardenete-Suriol, M.: EXTREME: Combining the Ease of Management of Multi-user Experimental Facilities and the Flexibility of Proof of Concept Testbeds. Proceedings of the First International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM '06), Barcelona, Spain (2006)

[10] Zheng, P., Ni, L. M.: EMWIN: Emulating a Mobile Wireless Network using a Wired Network. In Proceedings of the 5th ACM International Workshop on Wireless Mobile Multimedia (WOWMOM '02), pp. 64-71, Atlanta, GA (2002)

[11] Zheng, P., Ni, L. M.: EMPOWER: A Network Emulator for Wireline and Wireless Networks. Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003), Vol. 3, pp. (2003)

[12] Raychaudhuri, D., Seskar, I., Ott, M., Ganu, S., Ramachandran, K., Kremo, H., Siracusa, R., Liu, H., Singh, M.: Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-generation Wireless Network Protocols. Proceedings of IEEE Wireless Communications and Networking Conference, pp. 1664-1669, (2005)

[13] Zhou, J., Ji, Z., Varshney, M., Xu, Z., Yang, Y., Marina, M., Bagrodia, R.: WHYNET: a hybrid testbed for large-scale, heterogeneous and adaptive wireless networks. Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization, pp. 111-112, Los Angeles, CA (2006)

[14] Biswas, P. K., Phoha, S.: A Hybrid Infrastructure for Surveillance-based Sensor Network Experiments. Proceedings of the Second International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM '06), pp. 68-73, Barcelona, Spain (2006)

[15] Biswas, P. K., Poylisher, A., Chadha, R., Ghosh, A.: Hybrid Testbeds for QoS Management in Opaque MANETS, Proceedings of the 4th ACM SIGMOBILE/ICST International Wireless Internet Conference (WICON 2008) Hawaii, USA, Nov. 17-19 (2008)

[16] Chiang, C. J. et al.: Performance Analysis of DRAMA: A Distributed Policy-Based System for MANET Management. Proceedings of IEEE MILCOM, Washington DC (2006)

[17] Chadha, R., Cheng, H., Cheng, Y., Chiang, J., Ghetie, A., Levin, G., Tanna, H.: Policy-Based Mobile Ad Hoc Network Management, Policies for Distributed Systems and Networks, Proceedings of Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04), vol. 0, no. 0, pp. 35 (2004)

[18] The OPNET Simulator, http://www.opnet.com/